The Transactor

The Tech/News Journal For Commodore Computers Vol. 4.

$2.50

From The Editor's Desk (?)

Here it is! The first issue of The Transactor! Volume 4, Issue 1 marks a new chapter in the life of The Transactor since this is actually the 31st issue, but the first with a typeset interior, colour cover, and advertising.

Back in April of this year, Commodore Canada, the original publisher, decided to discontinue The Transactor in favour of the Commodore U.S. magazine. To fulfill subscription obligations, two more issues were assembled and the mail list program was prepared for its final output. It seemed that The Transactor would suffer the same fate as several other original PET publications, gobbled up by the "Mega-Mags".

Shortly after the last Transactor was shipped, the calls and letters started coming in. They ranged from critical to dissappointed. What could I say? I began answering explaining that, "the only hope for The Transactor was a chance for it to compete in the professional magazine industry, a battle The Transactor could not possibly withstand in its present form". Well, either the stars were in the right position that day, or my bio-wavelengths were being tuned in. The phone rang again. Frank Baillie of Canadian Micro Distributors (CMD) was on the other end.

"Karl, have you thought about taking on The Transactor as an independent publication?"

"Sure, but the time, work and costs involved would put me in the proverbial poor house. I'd need a sponsor."

"Well, you've got one. All we need is permission from Commodore."

The Transactor was alive again!

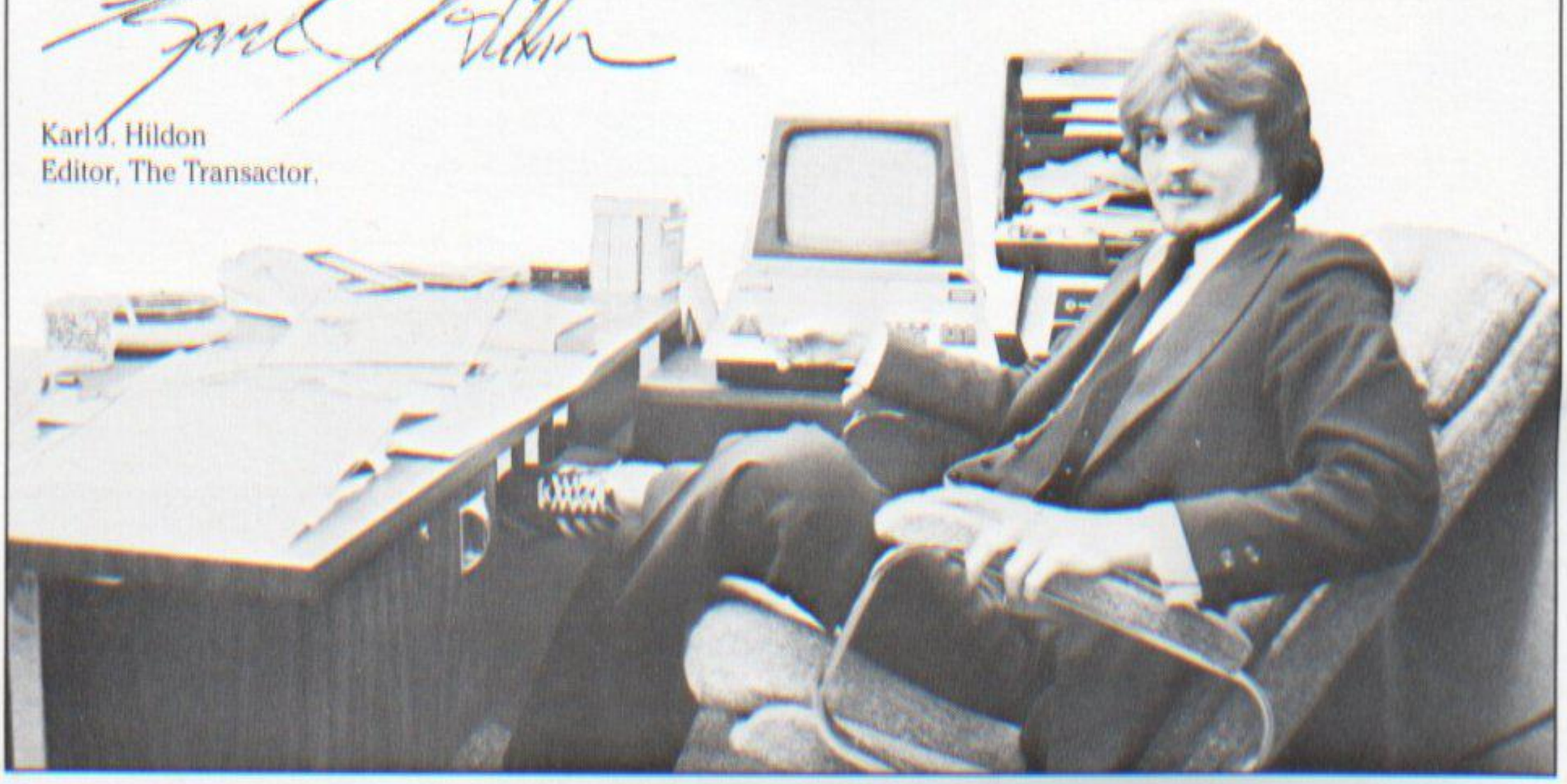
I'd like to take this opportunity to welcome back Transactor subscribers that have forwarded renewals. I've seen several familiar names come through our subscription department and we appreciate your support, without which there would be no Transactor. I'd also like to extend greetings to any new Transactor readers. New interest is a major objective for any magazine, and although we won't claim to be your single most desired computing resource, we do think that you'll find The Transactor informative, entertaining, and a valuable asset to your computer investment.

Like any magazine, we depend on revenue generated through advertising. But unlike most magazines, our sponsor is also in the business of selling micros, peripherals, and accessories. Consiously aware of the position a competetor may take when considering advertising under these circumstances, I'd like to point out right now that bias will in no way whatsoever influence any decision to accept advertising for any product. You'll see CMD ads in The Transactor, but no more so than any other magazine. The microcomputer industry seems to be one of the few that is unaffected by the economy. It would be a crime to suppress the exposure of any fine quality item that enhances the microcomputer market.

I welcome your comments, favourable or not, because your feedback is the key
to improvement, something I intend to make perpetual.

So, The Transactor has a new home, a new look, and a new future.
Once again, thank you for your support, I remain,

Karl J. Hildon
Editor, The Transactor.



The Transactor

Editorial	IFC	Tiny-Aid For VIC 20	39
Listings Legend	2	VIC RS–232 ASCII Driver	44
News and New Products	3	The Commodore 64:	
Bits and Pieces	8	A Preliminary Review	47
The WordPro Book Of Tricks	11	Architecture Map	50
The MANAGER Column	14	Memory Map	51
Disk Un–Assembler	17	Processor I/O Port & SID	58
Universal String Thing	22	CIA 1/2 Architecture	59
File Chain Tracer	24	Condensed Memory Map	60
Translation Arrays	27		
Filing It	30	Advertising Section	62
SuperPET Terminal Program	32	Advertising Index	64
APL & SuperPET Serial Port	34	Next Issue	64

Editor
Karl J. H. Hildon

Editorial Assistant
Elizabeth Baillie

Contributing Editors
Dave Berezowski
Jim Butterfield
Donna Green
Paul Higginbottom
Dave Hook
Eike Kaiser
Bill MacLean
John Stoveken

Production
Attic Typesetting Ltd.
With special thanks to
Nate Redmon, Phyllis Fast,
and all the others.

Printing
Squire & Carter Ltd.
With special thanks to
Ian Kernot

Cover Art
Jim Jones,
McLean–Waite Advertising

The Transactor is published bi–monthly, 6 times per year, by Canadian Micro Distributors, Limited. It is in no way connected with Commodore Business Machines Ltd. or Commodore Incorporated. Commodore and Commodore product names (PET, CBM, VIC, MAX, 64) are registered trademarks of Commodore Inc.

Volume 4 Subscriptions: Canada $15 Cdn
U.S.A. $15 US.
All other $18 US.

Second Class Mail
Permit Pending

Back issues are still available for Volumes 1, 2, & 3. Best of Volume 1: $10 Cdn., U.S.A. $10 US., all other $12 US. Best of Volume 2: $15 Cdn., U.S.A $17 US., all other $19 US. Volume 3: $10 Cdn., U.S.A $11 US., all other $13 US. Subtract $5 from total when ordering all 4 Volumes!

Volume 4 quantity orders: subtract 35% on orders of 10 or more.

Send all subscriptions to: The Transactor, Subscriptions Department, 500 Steeles Avenue, Milton, Ontario, Canada, L9T 3P7, 416 878 7277.

Want to advertise a product or service? Call or write for more information.

Editorial contributions are always welcome and will appear in the issue immediately following receipt. Preferred media is 2031, 4040, 8050, or 8250 diskettes with WordPro, WordCraft, Superscript, or SEQ text files. Program listings over 25 lines should be provided on disk or tape. Manuscripts should be typewritten, double spaced, with special characters or formats clearly marked. Photos of authors or equipment, and illustrations will be included with articles depending on quality. Diskettes, tapes and photos will be returned on request.

All material accepted becomes the property of The Transactor until it is published. Once released, Authors may re–submit articles to other publications at their own discretion. Other magazines, of any nature, are invited to copy material from The Transactor, provided that credit is given to the Author (when applicable) AND The Transactor.

The opinions expressed in contributed articles are not necessarily those of The Transactor. Although accuracy is a major objective, The Transactor cannot assume liability for errors in articles or programs.

Program Listings in The Transactor

All programs listed in The Transactor will appear as they would on your screen in Upper/Lower case mode. Many programs will contain reverse video characters that represent cursor movements, colours, or function keys. These will also be shown exactly as they would appear on your screen, but they're listed here for reference. (Editor's Note: Some characters could not be typeset in time for this list. They will appear on this page next issue)

Occasionally programs will contain lines that show consecutive spaces. Often the number of spaces you insert will not be critical to correct operation of the program. When it is, the required number of spaces will be shown. For example:

print" flush right" – would be shown as – print" [space10]flush right"

Cursor Characters For PET / CBM / VIC / 64

Down – q
Up – Q
Right –]
Left – [Lft]
RVS – r
RVS Off – R

Insert – T
Delete – t
Clear Scrn – S
Home – s
STOP – c

Colour Characters For VIC / 64

Black – P
White – e
Red – £
Cyan – [Cyn]
Purple – [Pur]
Green – ↑
Blue – ←
Yellow – [Yel]

Orange – A
Brown – U
Light Red – V
Grey 1 – W
Grey 2 – X
Light Green – Y
Light Blue – Z
Grey 3 – [Gr3]

Function Keys For VIC / 64

F1 – E
F2 – I
F3 – F
F4 – J

F5 – G
F6 – K
F7 – H
F8 – L

News and New Products

Do you have a new product or service? Just send all the details to us and we'll see that it's published in this section of the next Transactor. Ed.

Standardized Educational Software

This summer, about 30 Boards of Ed. have been working on a collection of over 1000 programs in an attempt to standardize educational software. On completion of this mammoth task, all of these programs will work on virtually every Commodore machine with the exception of the early BASIC 1.0 PETs, the VIC–20, and the new 'P' and 'B' series computers. BASIC 1.0 was excluded for obvious reasons, the VIC–20 due to its 22 column screen size, and the 'P' and 'B' series because of new internal architecture.

Each program has been fitted with about 6K of "overhead" subroutines. Pre–program code will test what machine the program is being run on. That means each program can be loaded into either the 2001 (BASIC 2.0), 4032, 8032, 8096, SuperPET (in CBM mode), or even the new Commodore 64, and will run without modification. Standard keyboard input and screen formatting routines have also been included to give the software a more uniform appearance.

Of course memory size will still restrict the use of larger programs, but most machines in Canadian schools have 32K. 80 column machines will require a screen modification program that converts to 40 column output. This will be available on a utilities disk along with copies of the "overhead" modules used in the standardization process. Other utilities will also be included to make programming life easier.

The complete library should be available from school boards across Canada by October 1st or earlier. For $10 you get a diskette, about 20 programs, and a Manager file containing details of the programs. A master catalogue disk can also be obtained for descriptions on the entire collection, an estimated 50 to 70 diskettes!

Commodore will be distributing the diskettes to their dealers sometime around November 1st, but no procedure details have been released. Possibly they'll be free for schools supplying their own diskettes?

Three New Commodore Info Sources

STRICTLY COMMODORE is a bi–monthly, software–oriented publication that is entirely devoted to VIC–20, PET, CBM, and SuperPET owners. Product reviews, programming tips, and other factual material is covered. Yearly subscriptions are $20 in Canada, $18 US. dollars for stateside subscribers, and $25 US. dollars in all other countries. Sample issues are $2 in Canada and the U.S., $3 elsewhere. Write to:

Strictly Commodore Inc.
Subscription Dept.
47 Coachwood Place N.W.
Calgary, Alberta
T3H 1E1

Strictly Commodore Inc. is also planning another publication. SUPER 64 will cover material on the new Commodore 64 and MAX, exclusively. Rates: $15 US. dollars in Canada and the U.S., $20 US. elswhere. Samples: $4 Canada and U.S., $5 for other countries. S.C. Inc. asks that all cheques and m/o's be in U.S. dollars only.

Editor' Note : So far I've only seen Volume 1/Issue 1 of STRICTLY COMMODORE. It came on 8 1/2 by 11's folded and stapled at the centre for 36, 5 1/2 by 8 1/2 inch pages. The programs were documented nicely with descriptions of important variables and conversions for other machines, however, their reproduction quality can only get better. Most of the mag is geared for beginners, which they openly admit, but their intentions are admirable with good growth potential.

PET PAPER and Midnight Merger

The PAPER (originally The PET PAPER), one of the oldest independent publications supporting Commodore computers is merging with another independent resource for Commodore users, the **Midnight Software Gazette**, beginning with the October, 1982 issue.

To celebrate the merger, a contest is underway to rename the merged magazines. The person suggesting the best name will receive a free VIC-20 courtesy of Computer Country of Springfield, IL. Judging will be by the editors, and in case of ties, by the readers of the combined magazine. Entries must be received by November 1st at the address below.

During the past two years, Midnite has become a widely respected resource in 41 states and 11 countries. Specializing in brief independent reviews of products for Commodore computers, it has received praise from nearly every magazine supporting CBM products. Its current issue is a 300+ page $10 book, with advance committments for 10,000 copies.

The PAPER has traditionally been a strong source of articles and tutorials for users of CBM equipment, with excellent series on such topics as first steps in machine language, as well as extended reviews of important products.

Subscriptions to the combined magazine are $20 US. or $25 CDN in North America for 6 bi-monthly issues. Overseas subscriptions are $45 US. Each issue will be professionally printed and securely shipped in envelopes via first class mail.

Send subscriptions to:

Midnite Software Gazette
635 Maple
Mt. Zion, IL
62549
217 864 5320

Commander

The first issue of **Commander** is slated for December 1st, 1982. Like The Transactor, it will be devoted to information on Commodore products only. **Commander** will be published monthly by Micro Systems Specialties, Tacoma, Washington.

Subscriptions are $22 in the U.S.A., $26 in Canada and Mexico, all other countries $37 surface or $54 air mail (I assume all figures are in US. dollars). Send all orders to:

Commander
P.O. Box 98827
Tacoma, Washington
98498
206 565 6818

Or call toll free in the U.S., (except WA, HI & AK) 1-800-426-1830 with your Visa, Master Card or American Express number. Subscribe before the premier issue and receive a $4 discount.

Editor's Note : Since a Commander has not yet been released, I can't give you an opinion. However, if what their promotional material says is true, it sounds like a very professional effort that should be considered.

Commodore Computing

This one is relatively new and worth mentioning. Without question, it's the best from the U.K. **Commodore Computing** was originally Commodore U.K.'s club newsletter (known to some as "CPUCN"). Commodore sold publication rights to Nick Hampshire publications, and Pete Gerrard, also a Commodore vet, is producing some super work. I've seen two excellent issues so far.

Subscription rates for 10 issues are 12.5 British pounds in the U.K. and 15 pounds overseas. You'll have to work that out in Canadian or U.S. dollars before sending. Air mail rates are available on request. Write or subscribe using:

Commodore Computing
Subscription Manager
MAGSUB
Oakfield House
Perrymount Road
Haywards Heath
Sussex RH16 3DH

VIC Computing

The only publication I know of devoted solely to the VIC-20. I imagine as Commodore brings out new home entries they'll cover these too. VIC Computing is not related to the above mentioned publication, but I believe they are closely associated with (if not owned by) Microcomputer Printout, a more general computing tabloid.

Annual subscriptions 6 pounds in the U.K., 9 in Europe and 16 overseas. Once again you'll have to re-calculate for Canadian and U.S. dollars. Their address:

VIC Computing
39-41 North Road
London, England
N7.9DP

Toronto Typesetter Connects To PET/CBMs

Attic Typesetting of Toronto have been in the typesetting business since 1973. Over the years, co-founder Nate Redmon has been experimenting with his Quadex 500 typesetting system and several microcomputers including Apple, Radio Shack, and just recently, Commodore. Data generated by the computer can be sent directly into the Quadex front end system. From there it's sent to the typesetting machine. Result: the infinite quality line printer!

The connection is a standard RS-232 line at 9600 baud. Using a SuperPET with Raymond Li's RS-232 terminal program, sequential files generated by WordPro for every article you see in The Transactor were transfered to the typesetter.

Even program listings can be typeset. Programs are simply LISTed to disk and sent like any other SEQ file. Cursor control characters can also be typeset, but only as they appear in Upper/Lower case mode. The major advantage, however, is the possibility of keyboarding error is eliminated.

Once in the editor, a spell checking program with over 200,000 words checks your text in seconds.

With all of the inherent capabilities, plus some custom programs written by Mr. Redmon for the Quadex, Attic offers a text processing service that's unparalleled by "off-the-line" typesetting equipment. If there's enough demand, Nate may consider installing a modem so that text can be sent remotely.

For more information contact:

Nate Redmon
Attic Typesetting Ltd.
5453 Yonge Street
Willowdale, Ontario
416 221 8495

Leading Edge Introduces Low-Cost 12" Monitor.

The newest addition to the Leading Edge Products' line of BMC monitors is the BM-12EN; a hi-res, non-glare, green screen 12" model in high demand for use by certain market segments such as professional, educational and specialized small businesses.

A 20 MHz bandwith gives the BM-12EN a sharper, more precise image, and the non-glare screen reduces eye strain substantially over prolonged periods. The unit comes with a one year repair/replacement warranty at a dealer cost of $99.00 U.S.

Leading Edge is the exclusive U.S. distributor of the BM-12EN, as well as the "Mean Green" BM-12AU, and the color composite BM-1400CL. All monitors are competitively priced and dealer inquiries are welcome. Contact:

Leading Edge Products
225 Turnpike Street
Canton, MA.
02021
Toll free : 1 800 343 6833
In MA & Can.: 1 617 828 8150
Telex : 951 – 624

On August 10, 1982, Leading Edge announced a one year, end-user parts and labor warranty for their complete line of C.Itoh Prowriters, F-10 Starwriters, and Printmasters. Con-

tact your nearest Leading Edge dealer for details.

Leading Edge Products is the largest marketer of microcomputer products in the U.S. employing over 150 people in 10 warehouses across the country

Remote Switched Isolators by Electronic Specialists

Remotely Switched Isolators announced by Electronic Specialists are the newest addition to their patented Isolator filter/suppressor line of interference control products. Available on all Isolator models, the remote AC power control switch can be mounted near the equipment operator for convenience and system cable neatness

Microcomputer applications include laboratory, hospital, industrial and personal systems having a need to conveniently control power to the operating system from a central location

Total Remote Switched Isolator load capacity is 1875 watts maximum with each socket capable of handling a 1 KW load. Remote switching available on models ranging from single-filtered models to quad-filtered magnum models. Prices start at $79.95

For more information contact:

Frank Stifler
Electronic Specialists, Inc.
171 South Main Street
Natick, MA.
01760
617 655 1532

IEEE For VIC-20 And Commodore 64

Richvale Telecommunications of Richmond Hill, Ontario, are now marketing the V-LINK, an IEEE interface for the VIC. The package consists of a cartridge that plugs into the cartridge slot on the back. On power-up, with the cartridge in place, the VIC Parallel User Port is configured as the IEEE port. The unit also includes BASIC 4.0 commands and can be ordered with additional RAM for expanding the memory of your VIC-20.

Richvale has a variety of options for the User Port IEEE: one is simply a connector that converts it to an IEEE card-edge (eg. for use with PET to IEEE cables), another is essentially a VIC to IEEE cable for daisy-chaining to other IEEE to IEEE cables: a third type of cable allows interfacing to printers such as Centronics and Diablo.

VIC owners now have easy access to IEEE peripherals of all sorts, including Commodore, Hewlett-Packard, and others. IEEE device owners will find the VIC/V-LINK combination an attractive controller for such units as IEEE oscilloscopes and spectrum analysers.

Just recently, V-LINK for the Commodore 64 was perfected. Due to the sophisticated nature of the 64, this unit is much tidier than it's predecessor. One connector does everything! On one side is a cartridge slot connector, on the other an IEEE card-edge connector. In between is a 4K ROM that contains the IEEE routines and also gives you BASIC 4.0 commands and a machine language monitor. No need for RAM expansion on this one; the 64 has all it can handle.

For details on prices, etc., contact

Peter Smith / Howard Hunt
Richvale Telecommunications
10610 Bayview Avenue, Unit 18
Richmond Hill, Ontario
L4C 3N8
416 884 4165

New Versions of SuperPET Languages

Commodore Canada has sent new versions of all the SuperPET language processors to all Canadian dealers. Commodore U.S. may be handling them differently, but I assume they should be available from any dealer by the time you read this notice. Cobol will also be in that shipment and, as announced, is free of charge to SuperPET owners and will be included with new SPET orders.

The Assembler and Editor programs are reportedly unchanged so your diskettes that contain either of these will not need updating.

A note on MicroFORTRAN. A recent discovery showed that FORTRAN does not like disk associated commands and statements that assume drive 0. To avoid potential errors, always include the drive number. This may also be true for some of the other languages, but hasn't been tested. Possibly this bug will be fixed in the new version.

Two New Assemblers From French Silk Smoothware

The Assembler for the Commodore VIC–20 is a machine language development system consisting of three programs: The Editor, The Assembler and The Loader. The package will function on all VIC configurations, including the 5K cassette–based system. The user–friendly design allows easy creation of machine language programs and BASIC–callable subroutines. A powerful two–pass symbolic assembler interprets algebraic expressions, decimal, hex, and literal data. It accepts programs from the source library created by The Editor and it creates object modules for use by The Loader. The ability to load and link multiple modules gives the VIC owner the opportunity to develope very large and powerful machine language programs (eg. a 4K program on the 3583 byte VIC). The user's manual is professionally written, clear and complete. The package is available on cassette for $24.95 plus $1.00 postage & handling, or on diskette for $29.95 plus $1.00 p&h.

Develope–64 for the Commodore 64 is a machine language development system consisting of an editor/assembler and a loader. With the editor/assembler, machine language programs may be created, modified, saved to the source library and assembled. Load modules are created for input by the loader and listings are produced containing source and generated machine language in either hex or decimal. This powerful two–pass symbolic assembler interprets algebraic expressions and a variety of data formats. Completely compatible with The Assembler for the VIC–20, Develope–64 can accept statements up to 78 characters in length and labels up to 73 characters long. This product is "human–engineered" for maximum user–friendliness. The manual is professionally written, clear and complete. Develope–64 is available on cassette for $34.95 plus $1.00 p&h, or on diskette for $39.95 plus $1.00 p&h.

Need more? Contact:

French Silk Smoothware
PO. Box 207
Cannon Falls, Minnesota
55009
507 263 4821

Toronto PET Users Group Now 2000 Members Strong!

TPUG started in 1978 with a mere 30 members. Chris Bennett, club secretary, informs me that this month they passed the 2000 member mark. They have regular monthly meetings at two locations in Toronto, special interest groups, their own newsletter (The TORPET), and a disk library with almost 50 diskettes!

Memberships are worth every nickel! They get you The TORPET and full access to the library.

Canadian Associate Members :	$20
U.S. Associate Members :	$30 US.
Overseas Associate Members :	$30 US.
Canadian Regular Members :	$30
Canadian Student Members :	$20

For more information contact (after business hours):

Chris Bennett
381 Lawrence Ave West
Toronto, Ontario
M5M 1B9
416 782 9252

Or use your computer to call the TPUG Bulletin Board at 416 223 2625. (7 PM. to 9 AM., Mon. through Sat. – All day Sun.)

Bits and Pieces

Got some interesting bit of info you'd like to share?...a POKE, a screen dazzler, a bug, or some other anomaly? Send it in! We'll ge glad to print it! Ed.

Optical Illusion

This neat little machine language program was written by Dave Berezowski at Commodore Canada. It doesn't do very much except create a rather interesting looking screen. The program will work on 40 or 80 column machines but the 80 column seemed to be the most impressive.

```
033c    ldx   #$00
033e    inc   $8000, x    ;vic users must subst.
0341    inx                 screen address
0342    bne   $fa
0344    inc   $033f
0347    jmp   $e455       ;for BASIC 4.0 users
0347    jmp   $e62e       ;for BASIC 2.0 users
0347    jmp   $eabf       ;for VIC-20 users
```

As you can see, the routine is interrupt driven which means you'll need to POKE the interrupt vector to get it going.

```
poke 144, 60 : poke 145, 3
```

After servicing this code, the normal interrupt routines are executed which means you'll still see the cursor. You can even edit (and RUN) BASIC while this is running, just don't try to use the cassette buffer that it lives in or whammo! Try moving the cursor around the "affected area".

Notice that the program is self modifying, a practice that is OK for small programs but should be avoided like the plague in larger ones. Self-modifying software is the worst for debugging and finding out the hard way is not fun.

Vic users could also get this going without too much difficulty (maybe even with colour?). Just substitute the PET/CBM screen start address ($8000 in the second line) with the start address of the screen in your particular Vic, one of two possibilities, $1E00 normally or $1000 with some memory expansion units. To engage it. . .

```
poke 788, 60 : poke 789, 3
```

For BASIC 4.0 users, just type in this loader. Others will need to change just the last two DATA elements and the interrupt vector POKEs.

```
10  for j=828 to 841 : read x : poke j, x : next
20  data  162,   0, 254,   0, 128, 232, 208, 250
30  data  238,  63,   3,  76,  85, 228
```

One last note. . .don't try to include the interrupt vector POKEs in the above program. Chances are your machine will crash because before both POKEs get executed, an interrupt occurs somewhere in-between.

Selective Directory

Ever been searching through your diskettes for a program and found yourself sifting through SEQ and REL filenames that just seem to get in the way? Or how 'bout the opposite. . .when you're looking for an SEQ or REL filename that's lost in diskettes full of programs. Well..here's a quick way around it.

```
LOAD "$0:*=PRG", 8
```

When finished, LIST will display all PRG files from the directory. It would stand to reason that matching type filenames would appear for both directories if the drive number were omitted, but such is not the case. If you leave out the drive number the disk only returns filenames from the last drive used.

Mysteriously, DLOAD won't work the same way. You must use the LOAD command followed by ',8'. Any file type can be selected though. Merely substitute PRG for SEQ, REL or USR.

Another variation. . .subtitute the * for filename patterns. This has been discussed before, but now you can look for filenames that match a pattern and are also of a particular type. . .

```
LOAD "$1:B*=SEQ", 8
```

. . .would load a directory of all sequential files on drive 1 that start with 'B'.

A Most Welcome Error Message?

Never thought you'd see the day an error message would be pleasant, did you? Well today is the day! Just turn on your machine, hit HOME and RETURN. Too bad you can only get it when the machine is empty!

Quick File Reader

This three–liner will read just about any SEQ file. It's not very sophisticated but when you just want to "take a boo" at a file, it can be typed in quickly and isn't too hard to memorize. The RVS will help to spot any trailing spaces.

```
10  dopen#8, "some file"
20  input#8, a$ : ?" "; a$ : if st=64 then dclose : end
30  goto 20
```

For REL files, simply change the IF statement in line 20 to:

```
if st=64 and ds=50 then. . .
```

The Dreaded Illegal Quantity

Sometimes you want to read files one byte at a time. A routine much like the one above might be used, only the INPUT# would be replaced by a GET#. There's just one minor gotcha. It seems that when a byte value of zero is retrieved by GET#, the string variable slated to receive it is set to a null string, not CHR$(0).

The most common occurence of byte–by–byte reading is with PRG files from disk. Program files contain lots of these zeroes, at least one per line of BASIC (end–of–line markers). Usually a program to read the PRG file is set up like this:

```
10  open 8, 8, 8, "some prg file,p,r"
20  get#8, a$ : print a$, asc(a$) : if st=64 then close 8 : end
30  goto 20
```

The problem is that when a zero is read into A$, the ASC(function cannot cope will a null string and bombs out with ?ILLEGAL QUANTITY ERROR. The solution? You could add an extra IF statement after the GET#, for example:

```
if a$="" then a$ = chr$(0)
```

. . .but that would mean an extra line for the PRINT statement and the following IF. . .rather clumsy. Keep things tidy with:

```
print a$, asc (a$ + chr$(0))
```

The ASC(function returns the ASCII value of the first character of A$. If A$ starts with a valid character, then adding CHR$(0) will make no difference. If not, then CHR$(0) will be added to the null string and a "0" will be printed rather than the dreaded illegal quantity error.

The Mysterious Extra Records

Those of you familiar with the Relative Record system will know that the end of a relative file is flagged by the ?RECORD NOT PRESENT error, DS=50. However, the last record used for data is not necessarily the last record of the file.

As relative files get bigger, the DOS formats additional sectors by filling them with "empty records". An empty

record starts with a CHR$(255) followed by CHR$(0)'s to the end of the record which is determined by the record length. This formatting process occurs when data is written to a record that will require more disk space than has been allocated to the file so far.

Each 256 byte sector can contain 254 bytes of data (the other 2 are used by the DOS). Let's take an example record length of 127, thus 2 records fit exactly into 1 sector. Imagine that 2 complete records have already been written to the file. Upon writing a third record, the DOS must format another sector. Two empty records are written, but the first will be replaced by the data of our third record. Closing the file causes our third record and the one empty record to be stored on the diskette.

Re-opening the file is no problem, but how do we find the next available space for writing a new record? Athough our fourth record is empty, a RECORD#lf, 4 will NOT produce a ?RECORD NOT PRESENT error and the CHR$(255) could successfully be retrieved and mistaken for valid information. Therefore, we must test the first character of the record for CHR$(255). An INPUT# of this record will result in a string of length 1, so a combination of the two conditions might be appropriate. However, INPUT#ing live records of length greater than 80 will produce ?STRING TOO LONG error, so GET# must be used in combination with an ST test:

```
1000  rem *** find next available record ***
1010  record# (lf), (rn)          :rem rn = record number
1020  get#lf, a$                  :rem get 1st char
1030  if ds = 50 then return
1040  if a$ = chr$(255) and st = 64 then return
1050  rn = rn + 1 : goto 1010
```

This subroutine will search forward from wherever you set RN initially. It stops when either a ?RECORD NOT PRESENT occurs or when an empty record is found. For larger files, you might consider starting at the end of the file an work backwards, but you'll need to find the first live record and then move the record pointer one forward.

In summary, relying on RECORD NOT PRESENT is not good enough. Although it will insure an empty record every time, it will eventually leave you with wasted disk space. Often the first record of the file is used to store a "greatest record number used" variable which is updated on closing and read back on opening. Although this is probably the cleanest approach, it will only return new record numbers. Any records that have been deleted by writing a single CHR$(255) must be found with a subroutine like above. Possibly a combination of both these techniques will produce a more efficient filing system.

The WordPro Book Of Tricks

Donna Green
Commodore Canada

The WordPro Book Of Tricks is a regular column by Commodore Canada's wordprocessing consultant, Donna M. Green.

Quick-LOAD – For "Plus" Versions Of WordPro

As you know it is a simple, quick matter to turn your Commodore computer into a word processor, once the chip has been inserted inside the computer.

After pressing Shift + Run/Stop which loads the program – several questions regarding the printer and device numbers appear on the screen. To skip over these questions, simply press:

CONTROL — and the Status Line will appear!

(Certain default values will be selected - "S" for NEC Spinwriter, "116" lines available – or whatever the maximum is on your WordPro version, "4" for Printer Device Number, "8" for Disk Drive Device Number.)

This will not only save time when starting up, but if you have a Spinwriter printer, you'll also be ready to print! Inputting, editing, and disk use is now possible; however, it is important to be aware that if you are intending to print using any other printer at this time, you will have to do another quick reset to select the proper printer.

What if you decide to print on a CBM printer after this quick start up? You can easily reset the system with the following command:

CONTROL then Shift + Run/Stop

You will then be asked the usual questions; for example, when it asks what printer, select "C" for CBM. If there is no change to the remaining questions, you may "return" through them until the control line is reached.

Using this quick reset, you can also change the number of lines in main text (from the maximum – which is the default value), to a smaller number if you wish.

If the "number of lines" is the only item to be changed, simply type in the new number, then press "CONTROL" to bypass the remaining questions.

Exit To BASIC (For "PLUS" Versions)

CONTROL + SHIFT + Q(QUIT)

This command will take you out of WordPro and back to BASIC without turning off your computer.

The Backslash Key – "\"

1. Make Use of "Shifted Mode" – Rather than the "Shift Lock" Key

To enter "Shifted Mode" – press the "\" key.

This will highlight the "S" on the status line and all letters will be typed as capitals – but numbers will stay as numbers. Therefore, to type a dollar sign for example, use the SHIFT key and number four. To exit from "Shifted Mode" press "\" again.

A Note on the Shift Lock Key : If the "shift lock" key is used for capitals or upper case characters, spaces will appear as solid lines on the screen; ie:

This__is__an__exampie.

These are actually "required spaces" used to keep items such as dates or names all joined together and to treat them as one word; this way they won't be split at the end of a line. When more characters than a line will hold are typed in this manner a "Format Error" message could result because the system cannot find a "space" to use as a line ending break point. For this reason it is better to use the "shifted mode" at all times, rather than the "shift lock" key.

2. Turn on the Beeper (Plus versions)

CONTROL + \ (same turns beeper off)

The Status Line will indicate either "Sound On" or "Sound Off". The beeper will sound when you switch text areas, recall, memorize, update, or finish printing a document.

3. Shortcut for Recalling – Using Backslash Key "\"

You are also probably aware of the command to recall when the directory is on the screen. Before giving the command, move the cursor to the "file name" and while pointing to it with the cursor either in the space preceding, or on the first character of the name – press:

SHIFT + CLR/HOME + R(RECALL) + \ + RETURN

4. Make Use of Comment Lines on Every Page for Faster Recalling

Make use of a "comment line" as the first line of your document (checkmark cm colon + name of file) and keep a copy of the file name here. Then, when updating or Memorizing a document, go to this comment line and have the cursor on the first character of the file name. Press \, and the file name will appear on the status line, rather than retyping the name.

5. Quick Recall of File with Similar Name

To take advantage of the "\" shortcut (as in #3 above) without bringing the directory to the screen, recall a document with a similar name the following way:

Cursor to the first character of the file name (on line 1 in the "comment" section), then press:

SHIFT + CLR/HOME + R(RECALL) + \
INST/DEL KEY + NEW NO. + RETURN

Example: If "report page 3" was the name indicated in the "comment line", and "report page 1" was required, recall using "\" as above then press INST/DEL to delete the "3" then type "1" and return.

In a linked document, the "nx" command which is always the bottom line, can be used just like the "cm" command (which is usually the first line). The cursor can point to this name, and the backslash key can then be used to display the "file name" for recalling purposes.

FOR OTHER "QUICK RECALL" TIPS. . .

1. Quick Recall/Scroll Through Linked Files

This is a great time saver and a very helpful aid when working with long linked documents:

SHIFT + CLR/HOME + R + CLR/HOME + RETURN

When the CLR/HOME key is pressed the second time in this command, WordPro automatically searches out the "nx" command and displays the file name contained there on the Status Line.

2. Wild Card for Recalling – The Asterisk (*)

Perhaps you have forgotten exactly how a "file name" was spelled on the directory but you know it started with "st". To recall the first document on the disk that begins with "st", give the following command:

SHIFT + CLR/HOME + R(RECALL) + st* + RETURN

This will find the first document that begins with "st" and recall it to the screen.

3. Recall Both Directories With One Command

CONTROL + 2 + RETURN

The status line will prompt "Selective: __". By typing "2" at this time it will bring both directories to the screen, Disk Drive #1 first, and below it Disk Drive #0. Obviously, a disk must be in both drives for this to work. (This will therefore not work on the 2031 single disk drive.)

4. Recall Selectively from Directory

If there are disks in both Disk Drives "0" and "1", the following command can be given to quickly view all the files that begin with given characters:

CONTROL + 2 + st* + RETURN

This will bring to the screen a listing of only the files that begin with "st" from both Drive #1 and Drive #0. To recall a specific file, move cursor to file name and recall using the backslash key as described above.

Quick Cursor Movements

1. Scroll Up & Down Quickly

As you already know, if you wish to scroll the cursor quickly to the bottom of the screen, press:

CONTROL + CURSOR ARROW DOWN

To quickly scroll back up the screen, press either HOME/HOME, or

CONTROL + CURSOR ARROW UP

2. To Jump to Bottom of Text

However, there is another way to reach the bottom of your document without scrolling through the text on the screen, and possibly going too far – press:

CONTROL + TAB KEY (OR BACK ARROW, OR UP ARROW)

The cursor will quickly jump down to the last line of typing in your document, and stop at column 80.

3. Cursor Forward – To Resume Inputting

To move the cursor forward from this point (column 80) – press:

CURSOR RIGHT KEY

This will move the cursor forward one position (past column 80) to "column one" on the line below – exactly where you would probably want to continue typing!

Also, when a tab is set (control + s) at column 80 — by pressing the TAB key, the cursor can quickly be moved to the right side of the screen.

Using these faster ways of moving the cursor around the screen will be very helpful in editing text and making more efficient use of the system.

4. Return Carriage Without Deleting Rest of Line

When the cursor is in the middle of a sentence, to return to the beginning of the line below without erasing information by accident – press:

SHIFT + RETURN

To quickly move the cursor up one line to column 80 or the end of the preceding line – press:

SHIFT + CURSOR ARROW LEFT

Of course, since the cursor cannot go left of column 1, it will move instead to column 80 of the line above!

That's all for now. More next issue!

The MANAGER Column

John Stoveken
Milton, Ont.

The Manager Column is a regular feature of The Transactor. If you have any tips for other Manager users, send them in and we'll include them in the next issue. Ed.

The MANAGER software system has become one of the more popular database packages for the 80 column Commodore computers. Now for some programmers goodies. . .

The following listings illustrate some of the BASIC extensions that are contained in the MANAGER machine code.

String Input

x$=" " : \i, n, x$: x$=x$

This routine inputs a string into the called variable from logical file number "n". The input routine will bring in all ascii characters and terminate only on a carriage return (ascii 13). The x$=x$ is necessary to transfer the string from the reserved buffer into BASIC space. Maximum string length is 255.

Screen Dump

open 4, 4 : \d : close 4

This routine performs a screen dump of the current contents of the screen to the printer. It will perform a PET-ASCII conversion depending on the contents of memory address 22527; ie, poke22527, 14 = ASCII, poke22527, 12 = PET.

String Insert

a$="aaa" : b$="1234567890" : \m, a$, b$, n

This routine inserts the string a$ into b$ starting at position n. For example, if n = 3 the result of this operation would result with:

a$ = "aaa"
b$ = "12aaa67890"

Mode Toggle

\a

Simply toggles the PET from upper–lower case to uppercase–graphics mode, or vice versa.

Printer Output

\p, a$

Prints a string to logical file 4 (which is OPENed by The MANAGER on device 4) performing the same PET-ASCII conversion as in the \d command (dependent of contents of location 22527).

Toggle Auto-Repeat/STOP Key

\r

Simply toggles the auto-repeat, disabled STOP key.

Program Load

\l, "program"
This command will load a program from disk to memory. If the command \l; "name" is used then a machine code module (or .scr file) can be loaded into memory without mucking up the BASIC pointers.

Program Load & Execute

\x, "name"

This command will load and execute a BASIC program from disk. Its counterpart, \x; "name" will load a machine code module and 'sys' to the start address of that module.

These routines may be used in your own software if it is called from "The MANAGER" package. The simple route to call a program from The MANAGER is to modify the "MENU" program by adding these two lines.

```
146 print spc(15)"zee new program name"
166 if a$="Z" then \x, "new program"
```

MANAGER files may be viewed with a very simple program, that must be called from the MANAGER.

```
 0 ba=18432 : rem base address for machine code
10 print "input file name"; : l=16 : gosub 1000
20 na$ = x$
30 print "input drive"; : l=1 : gosub 1000
40 d = val (x$)
45 rem *** Open and read the file ***
50 dopen#1, (na$), d(d)
60 a$="" : \i, 1, a$ : a$=a$
61 rem inputs a string into the specified variable (a$) from
   the logical file specified (1)
70 if a$=chr$(255) then dclose : stop
71 rem character 255 indicates the end of a MANAGER
   relative file
80 print a$ : goto 50
1000 rem *** Input a string loop ***
1010 x = peek(198) : y = peek(216)
1020 x$="" : sys ba+57, y, x, l
1021 rem inputs a string of length l at the row and column
     specified by y and x into the variable x$
1030 return
```

If we wished to view a MANAGER file in sorted order, we must first scan the pointer file associated with the main file and use it to give us the records in the correct order. To do this, we will change lines 50 to 90 to permit this.

```
50 dopen#1, (na$), d(d)
60 np$=na$ + ".ptr" : dopen#2, (np$), d(d)
70 a$ ="" : \i, 2, a$ : a$=a$
72 rem this inputs a record from the pointer file in the
   "sorted" sequence
80 sx=st : rem store status of ptr file
```

The record position in the main file is "stored" in the last two bytes of the pointer (or .ind file), and can be decoded with this routine:

```
90 nn=(asc(right$(a$,2))+256*(asc(right$(a$,1))))/
   32768+1
100 record#1, (nn)
110 x$="" : \i, 1, x$ : x$=x$
120 print x$ : if sx=64 then dclose : stop
130 goto 70
```

More MANAGER Notes

Three sub-programs of The MANAGER system have the option of using "search criteria" to select records from a file. The three are: Global Update, Produce Sub-Files, and Report Generate.

Those of you that use The MANAGER will know that ALL conditions specified in the search criteria must be met before a record will be selected. Inotherwords, condition #1 must be true, AND condition #2 must be true, AND condition #3. . . through to condition #n. Even if one fails, the record is discarded and testing begins on the next record.

But suppose we'd like to select a record if only one of the search criteria were true. That is, condition #1 must be true, OR condition #2 must be true, OR #3, etc. For example, to produce a sub-file containing records that have "Ottawa" or "Toronto" in the City field would be impossible using The MANAGER as is. For this we need one simple modification.

The program you modify will depend on which function you plan to use it with. For Global Update the program name is "global", Produce Sub-Files is "finish" and Report Generate is "generate2". Fortunately the line number is the same in all cases. Load the appropriate program using a DLOAD command and LIST line 2370. You should see something like:

```
2370 ff=ff+fl : next : fl=1 : if ff<nf then fl=0 : ff=0
```

The number of fields to be tested is represented by the variable nf. The variable FF is a flag accumulator. Variable FL is set to 1 if a condition is true, and 0 if false. Therefore FF must be equal to NF for all conditions to be true. If FF is less than NF, FL and FF are set to zero which tells another part of the program to continue with the next record. To get the program to accept "any condition true", change line 2370 to:

```
2370 ff=ff+fl : next : fl=1 : if ff=0 then fl=0 : ff=0
```

A one byte change! Now the program will only skip to the next record if NONE of the conditions are met. Thus the search criteria function has been changed from and ANDing operation to an ORing operation.

You could take this one step further and replace the IF statement to a GOSUB. The subroutine might test for certain pairs of conditions to be true. Just set FL and FF appropriately before exiting.

This "new" program must be SAVEd under the same name so that the MANAGER menu program can load it (remember, this program can't be run unless the machine language subroutines are set up first). Rename the original version with:

```
rename d0,"global" to "global.and"
```

Alternately, it wouldn't be difficult to modify the menu program to call either version by choice.

Larger MANAGER Data Files

As you all know, REL files on the Commodore 8050 are limited to a maximum size of 180K. This is why The MANAGER can not fill an 8050 disk with just one file. But if you get the new DOS for the 8050, (available now or soon from CBM) the restriction is lifted. The same holds true for the new 8250 and the 9060/9090 hard disks.

To allow for larger files, two programs must be slightly modified. Logically, they are the Create option (filename "create") and the Manipulate Files option (filename "fileman"). Each program needs only one line changed. The lines are:

"create" : line 3070
"fileman" : line 13070

LIST the appropriate line and simply place a REM statement at the beginning. You could delete the line completely, but this way, if you ever want the line back again for use with the 8050, just remove the REM. Now DSAVE it back to the disk using replace, eg:

```
dsave "@create"
```

Disk Un–Assembler

Paul Higginbottom
Toronto, Ont.

You've all heard of a "disassembler". . . a utility for displaying machine code programs. But disassemblers are usually limited to just that, ie. display only. You can list the code, follow it around, and with some disassemblers like the one in Supermon, even make minor changes. But inserting code and major mods can be an awkward and tedious task.

Unless you have source code. Most machine language programs are made from source code. . . a file, generated by a programmer (using some sort of editor), that contains the start address of the program followed by variable definitions, machine language intructions, word and byte tables, plus everything else that goes into producing a chunk of machine code. An assembler is then used to convert source code into a machine language program. However, this is usually all you get; a file packed tightly with the hex instruction codes. Often the source is not made available by the programmer, and although a disassembler will undo the program into an intelligible listing, it can not reverse the process back to source code. For this we need an "Un–Assembler".

Small machine language programs can be "POKEd in" or hand–assembled without too much trouble. But larger programs usually require the aid of an assembler, or in most cases, a symbolic assembler. Symbolic assemblers allow the use of labels in source code. Labels can be any combination of letters and numbers, but must start with a letter. The CBM assembler allows a maximum of 6 characters. A label can be used anywhere there can be an address or an expression. Transfering execution is actually only possible by referencing a label. For example:

```
                JSR   OPNDSK
INPUT           JSR   GETCHR
                CMP   #CR
                BNE   INPUT
```

By using labels, one can write machine code without the need for calculating branch offsets, remembering storage location addresses or subroutine addresses, or the value of a particular character. Besides, they might all change as modifications are made.

A disassembler won't show you labels but rather the absolute values. JMPs and JSRs will be followed by hex addresses, branch intructions by their actual offsets, and load, store, compare intructions and the like, by a hex address or number. Although a disassembler could be used to create a load file for an editor, the result would be approaching useless. Every branch, JSR, and JMP address would have to be altered by hand to a label. The same label would have to be inserted at the lines containing the destination instruction. That's a lot of work.

Enter the Un–Assembler. This one was written by Paul

Higginbottom. It produces source code files compatible with the Commodore assembler editor. The beauty of Paul's program is that it works from disk. The program you wish to produce source code for does not have to be resident in memory which is a problem when the program lives in the same space as the Un-assembler. Labels for branches and jump type instructions are the only ones generated. Other instructions are left alone.

The un-assembler opens the machine language program on disk as a program file. Another write file is opened that will used to store the source listing. Three passes are made through the program file. On pass 1, the first two bytes that represent the load or start address of the program are retrieved. Then it runs right through the file to find the end of the program. This is important since only jumps and branches within the program itself are labeled. Jumps and branches outside of the program cannot possibly be given a label simply because the code at the destination does exist in this file.

Pass 2 finds all the jumps and branches that transfer execution to some point within the boundaries of the program. These entry points are "tagged" by the Un-assembler.

Pass 3 completes the job. The Un-assembler re-opens the program file and strips off the first two bytes. At this point it looks in the tag table to see if this spot is an entry point. If it is, a label is sent to the source code file that is a combination of the letters "AD" followed by the actual address of the next instruction. Now it retrieves this byte and disassembles it. When it determines the mode of the instruction, 0, 1 or 2 more bytes are retrieved. If this instruction was a branch or a jump to some destination within the program, the Un-assembler substitutes the offset or address with a label. The label will be "AD" followed by the address of the destination. The process continues until the entire program is un-assembled. Paul's program will automatically start a new file if the first gets too large for the Commodore editor.

There are a few peculiarities to watch for. First, the Un-assembler will try to decode everything. That means even .byte and .word tables will be treated as code. Byte tables are often used for things like command tables or error messages. Usually they contain ASCII characters and although some ASCII values have corresponding machine codes, it's unlikely that an entire table would be disassembled into sensible code. What you'll probably see in the resulting source code file is a number of .byt directives, separated occasionally by some rather silly looking disassembly. This silliness can be changed back to .byts, or left as is, according to preference.

Secondly, BIT instructions get special treatment. Often, a BIT instruction is used to "hide" a 1 or 2 byte code. Since this hidden code will probably mark an entry point, BIT instructions are placed on .byt lines. The Un-assembler doesn't attempt to distinguish real BIT instructions from one of these "code-hiders" so you'll have to decide whether to make changes to the source.

Also watch for programs that start at the beginning of BASIC text space. These will probably start with a SYS command, but the Un-assembler will still try to decode it.

Ultimately, whatever the Un-assembler produces can be re-assembled (using the Commodore assembler) into exactly the same program you started with. The labels it generates might be substituted for more meaningful names and, of course, it won't comment your listing. Chances are you're using this program because you can't get the original source. Although the Un-assembler won't give you a source listing as complete, you sure will get it quicker and with no questions asked.

```
100 poke52,peek(42) : poke53,peek(43)+27 : clr : poke1,peek(52) : poke2,peek(53)
110 ts=peek(52)+peek(53)*256 : for i=634to654 : read a : poke i, a : next : sys 639
115 data165, 251, 76, 34, 215, 160, 0, 152, 145, 1, 230, 1, 208, 2, 230, 2, 166, 2, 16, 244, 96
120 h2=634 : h4=55063 : al=251 : ah=252 : c1=1 : c2=2 : mh=256 : p=0 : mr=127 : mt=128 : ms=16
121 q=0 : h$="" : a$="" : b$="" : c4=4 : i=0 : a1=0 : as=0 : deffnl(p)=(p/mh-int(p/mh))*mh
130 as=0 : nl$=chr$(0) : a=0 : re=0 : ad=0 : deffnas(a)=((a>31)and(a<128))+(a>159)
140 lf=1000 : lc=0 : def fn rt(x)=peek(x+ts) : dim mn$(255), md(255)
150 print" Disk Un-Assembler" : print"by Paul Higginbottom
754 for i=1 to 149 : read a, a$, b : mn$(a)=a$ : md(a)=b : next
790 print : input"drive, program filename";dr$,f$ : if dr$<"0"or dr$>"1"then790
791 open1,8,15 : print#1,"i"+dr$ : f$=dr$+":"+f$
801 input"drive, source filename";td$,of$ : if len(of$)>12 then 801
802 of$=td$+":"+of$+"." : if td$<"0" or td$>"1" then 802
803 print#1,"i"+td$ : close 1 : open 2,8,0,f$ : if ds then print ds$ : close 2 : end
805 t=ti : gosub 3000 : nf=2 : nf$=of$+"1.s" : p=s-c1 : ss=0
809 open3,8,3,"@"+nf$+",s,w" : printnf$ : lc=0 : if dd then print ds$ : end
810 if p=s-c1 then print#3,"*=$"; : ad=s : gosub 9000 : print#3 : print#3,";"
815 lc=lc+1 : if lc<>lf then 830
816 nf$=of$+mid$(str$(nf),2)+".s" : print#3,";" : print#3,".fil "nf$
817 close 3 : nf=nf+1 : goto 809
830 p=p+c1 : gosub 2000 : mn=q : sa=p : gosub 4000 : print#3," ";
840 if mn=36 or mn=44 then print#3,";<this was a bit instruction>" : goto 845
841 if mn$(mn)<>"" then 850
845 print#3,".byt $"; : ad=mn : gosub 9010 : print#3 : goto 815
850 print#3,mn$(mn)" ";
860 on md(mn) goto870,900,930,960,990,1020,1050,1080,1110,1140,1170,1200,1230
870 gosub 2000 : print#3,"#$"; : ad=q : gosub 9010 : print#3 : p=p+c1 : goto 815
900 print#3,"a" : goto 815
930 print#3 : goto 815
960 gosub 2000 : print#3,"$"; : ad=q : gosub 9010 : print#3 : p=p+c1 : goto 815
990 gosub 2000 : ad=q : gosub 2000
1000 p$="$" : ad=ad+q*mh : if ad<s or ad>e then 1019
1001 if fnrt(ad-s) then p$="ad"
1019 print#3,p$; : gosub 9000 : print#3 : p=p+c2 : goto 815
1020 gosub 2000 : ad=q : gosub 2000
1025 p$="$" : ad=ad+q*mh : if ad<s or ad>e then 1030
1026 if fnrt(ad-s) then p$="ad"
1030 print#3,p$; : gosub 9000 : print#3,",x" : p=p+c2 : goto 815
1050 gosub 2000 : ad=q : gosub 2000
1051 p$="$" : ad=ad+q*mh : if ad<s or ad>e then 1065
1052 if fnrt(ad-s) then p$="ad"
1065 print#3,p$; : gosub 9000 : print#3,",y" : p=p+c2 : goto 815
1080 gosub 2000 : print#3,"($"; : ad=q : gosub 9010 : print#3,"),y"
1090 p=p+c1 : goto 815
1110 gosub 2000 : print#3,"($"; : ad=q : gosub 9010 : print#3,",x)"
1120 p=p+c1 : goto 815
1140 gosub 2000 : print#3,"$"; : ad=q : gosub 9010 : print#3,",x"
1150 p=p+c1 : goto 815
1170 gosub 2000 : print#3,"$"; : ad=q : gosub 9010 : print#3,",y"
1180 p=p+c1 : goto 815
1200 gosub 2000 : ad=p+q+(q>mr)*mh+c2 : p$="$" : if ad<s or ad>e then 1220
1211 if fnrt(ad-s) then p$="ad"
1220 print#3,p$; : gosub 9000 : print#3 : p=p+c1 : goto 815
```

```
1230 gosub 2000 : ad=q : gosub 2000
1231 print#3,"("; : p$="$" : ad=ad+q*mh : if ad<s or ad>e then 1240
1232 if fnrt(ad-s) then p$="ad"
1240 print#3,p$; : gosub 9000 : print#3,")" : p=p+c2 : goto 815
2000 get#2,a$ : q=asc(a$+nl$) : return
3000 print"  pass1" : get#2, a$, b$ : rem get start address
3010 s=asc(a$+nl$)+asc(b$+nl$)*mh : e=s
3021 get#2, a$ : e=e+1 : if st=0 then 3021
3022 gosub 9600 : p=s-1
3025 p=p+1 : gosub 2000 : n$=mn$(q) : n=md(q)
3026 on n gosub 3050, 4010, 4010, 3050, 3100, 3100, 3100, 3050, 3050, 3050, 3050, 3110, 3100
3030 if p<=e then 3025
3039 gosub 9600 : print"  pass2  " : return
3050 gosub 2000 : p=p+c1 : return
3100 gosub 2000 : ad=q : gosub 2000 : ad=ad+q*mh
3101 if ad>=s and ad<=e then poke ts+ad-s, c1
3102 p=p+c2 : return
3110 gosub 2000 : ad=p+q-(q>mr)*mh+c2
3111 if ad>=s and ad<=e then poke ts+ad-s, c1
3120 p=p+c1 : return
4000 ifp>e then print#3,";" : print#3,".end" : close 3 : close 2 : end
4001 if p<s or p>e then return
4005 if fnrt(p-s) then print#3,";" : print#3,"ad"; : ad=p : gosub 9000
4010 return
9000 poke al, fnl(ad) : poke ah, ad/mh : cmd3,; : sys h4 : return
9010 poke al, ad : cmd3,; : sys h2 : return
9600 close 2 : open 2, 8, 0, f$ : get#2, a$, a$ : return
10000 data   0,  brk,  3,   1,  ora,  9,   5,  ora,  4,   6,  asl,  4
10010 data   8,  php,  3,   9,  ora,  1,  10,  asl,  2,  13,  ora,  5
10020 data  14,  asl,  5,  16,  bpl, 12,  17,  ora,  8,  21,  ora, 10
10030 data  22,  asl, 10,  24,  clc,  3,  25,  ora,  7,  29,  ora,  6
10040 data  30,  asl,  6,  32,  jsr,  5,  33,  and,  9,  37,  and,  4
10050 data  38,  rol,  4,  40,  plp,  3,  41,  and,  1,  42,  rol,  2
10060 data  45,  and,  5,  46,  rol,  5,  48,  bmi, 12,  49,  and,  8
10070 data  53,  and, 10,  54,  rol, 10,  56,  sec,  3,  57,  and,  7
10080 data  61,  and,  6,  62,  rol,  6,  64,  rti,  3,  65,  eor,  9
10090 data  69,  eor,  4,  70,  lsr,  4,  72,  pha,  3,  73,  eor,  1
10100 data  74,  lsr,  2,  76,  jmp,  5,  77,  eor,  5,  78,  lsr,  5
10110 data  80,  bvc, 12,  81,  eor,  8,  85,  eor, 10,  86,  lsr, 10
10120 data  88,  cli,  3,  89,  eor,  7,  93,  eor,  6,  94,  lsr,  6
10130 data  96,  rts,  3,  97,  adc,  9, 101,  adc,  4, 102,  ror,  4
10140 data 104,  pla,  3, 105,  adc,  1, 106,  ror,  2, 108,  jmp, 13
10150 data 109,  adc,  5, 110,  ror,  5, 112,  bvs, 12, 113,  adc,  8
10160 data 117,  adc, 10, 118,  ror, 10, 120,  sei,  3, 121,  adc,  7
10170 data 125,  adc,  6, 126,  ror,  6, 129,  sta,  9, 132,  sty,  4
10180 data 133,  sta,  4, 134,  stx,  4, 136,  dey,  3, 138,  txa,  3
10190 data 140,  sty,  5, 141,  sta,  5, 142,  stx,  5, 144,  bcc, 12
10200 data 145,  sta,  8, 148,  sty, 10, 149,  sta, 10, 150,  stx, 11
10210 data 152,  tya,  3, 153,  sta,  7, 154,  txs,  3, 157,  sta,  6
10220 data 160,  ldy,  1, 161,  lda,  9, 162,  ldx,  1, 164,  ldy,  4
10230 data 165,  lda,  4, 166,  ldx,  4, 168,  tay,  3, 169,  lda,  1
10240 data 170,  tax,  3, 172,  ldy,  5, 173,  lda,  5, 174,  ldx,  5
```

```
10250 data 176,  bcs, 12, 177,  lda,  8, 180,  ldy, 10, 181,  lda, 10
10260 data 182,  ldx, 11, 184,  clv,  3, 185,  lda,  7, 186,  tsx,  3
10270 data 188,  ldy,  6, 189,  lda,  6, 190,  ldx,  7, 192,  cpy,  1
10280 data 193,  cmp,  9, 196,  cpy,  4, 197,  cmp,  4, 198,  dec,  4
10290 data 200,  iny,  3, 201,  cmp,  1, 202,  dex,  3, 204,  cpy,  5
10300 data 205,  cmp,  5, 206,  dec,  5, 208,  bne, 12, 209,  cmp,  8
10310 data 213,  cmp, 10, 214,  dec, 10, 216,  cld,  3, 217,  cmp,  7
10320 data 221,  cmp,  6, 222,  dec,  6, 224,  cpx,  1, 225,  sbc,  9
10330 data 228,  cpx,  4, 229,  sbc,  4, 230,  inc,  4, 232,  inx,  3
10340 data 233,  sbc,  1, 234,  nop,  3, 236,  cpx,  5, 237,  sbc,  5
10350 data 238,  inc,  5, 240,  beq, 12, 241,  sbc,  8, 245,  sbc, 10
10360 data 246,  inc, 10, 248,  sed,  3, 249,  sbc,  7, 253,  sbc,  6
10370 data 254,  inc,  6
```

Universal String Thing

Jim Butterfield
Toronto

This is not a complete String Thing*, just the INPUT# section. Quite simply, the program is an INPUT# utility that lifts some of the restraints imposed by the BASIC INPUT# command. It works on all versions of BASIC (except 1.0) for PETs and CBMs. If there's enough demand, a VIC String Thing will be released that will probably work on the Commodore 64 when it arrives.

Logical file number 1 must be used in the DOPEN or OPEN statement. Using the routine with any other file number will result in ?FILE NOT OPEN ERROR after the SYS call.

No buffer is required for the incoming data. The characters are built directly into the space allocated for the string variable. If the string resides in high RAM, they will be delivered there. If the string is declared in the program, the characters will land right in text space.

The string used for input must be the first variable seen by BASIC. Location 189 decimal is used to store the length of the input string. Input stops only on Carriage Return and EOI which occurs at the end of a file or relative records not terminated by a CR. Leading spaces, commas, colons and quotation marks are all accepted.

The size of the string can be used to govern the amount of input. For example, removing lines 110 and 120 from the program below leaves a$="abcdefghijklmnopq", with a length of 17. Now, strings longer than 17 can not be retrieved in whole. Instead, the first 17 characters are brought in and the remainder is received in subsequent calls. If the string is exactly 17 long followed by a Carriage Return, a subsequent input would be met by the CR and PEEK(189) would equal zero.Be careful though. In this case A$ will contain the previous input, not null string.

Using this routine as opposed to INPUT# can be especially helpful when disk space is at a premium. It can handle strings up to 255 characters long and there's no need for Carriage Returns.

* – The String Thing was originally a utility written by Bill MacLean of BMB Compuscience. It included string search, string overlay, and string input functions. For details on this program, see The Transactor, Volume 3, Issue 1. Copies are available from the Toronto PET Users Group.

```
 40 rem ***********************************
 50 rem **      string thing (universal)     **
 60 rem **          jim butterfield          **
 70 rem **        string used for input      **
 80 rem **        must be first variable     **
 90 rem ***********************************
100 a$="abcdefghijklmnopq"
110 a$=a$+a$+a$+a$+a$
120 a$=a$+a$+a$
130 rem above sets string for maximum (ie. 255)
200 data 160,   2, 177,  42, 153, 184,   0, 200, 192,   6
210 data 208, 246, 162,   1,  32, 198, 255
220 data  32, 228, 255, 201,  13, 240,  11, 164, 189, 145
230 data 187, 200, 132, 189, 196, 186, 208, 238,  76, 204, 255
250 for j=896 to 933 : read x : poke j, x : t=t+x : nextj
260 if t<>5767 then stop
400 dopen#1, "some SEQ file" ,d0
410 rem : next sys same as 'input#1,a$'
420 sys 896
425 rem : l=size of input (could be 0)
430 l=peek(189)
440 print left$(a$,l)
450 if st=0 goto 420
460 dclose
```

File Chain Tracer

Bill MacLean
Milton, Ont.

Think about it. Relative record lengths limited only by the capacity of your disk. Number of files open simultaneously limited only by FRE(0). All possible using this utility by Bill MacLean of Milton Ontario.

Every disk file (SEQ, PRG, or USR) is contructed from sectors that are linked together using "chain pointers". Each chain pointer consists of two bytes that represent the track and sector co-ordinates of the next block in the file. The pointer to the first block is stored in the directory beside the filename. Subsequent pointers are stored in the first two bytes (0 and 1) of each sector in the file. To indicate the last sector of a file, the track co-ordinate is set to zero (CBM disk units do not use track 0 for this reason), and the byte used for the sector co-ordinate is set to the last position used.

By reading all of these pointers into an array, a complete map can be built of any file on disk. Then, using a Direct Access channel, any part of any file can be read at any time with a 'block-read' command followed by the corresponding track and sector values stored in the array for that file. Positioning into the sector is done with the "buffer-pointer" command.

There are several ways of tracing these pointers to the end of a file. The trick, of course, is to get the first one out of the directory. One could OPEN the directory as an SEQ file, but searching through a long directory is slow in BASIC. Bill's program lets the DOS do that work in machine language. When a file is OPENed, the DOS must also know where the directory entry lies. These values are stored in DOS memory and can be retrieved with the "memory-read" and GET# commands. Once the first sector is found from the directory, the following sectors are "block-read" with the "U1" command, and the pointers are collected by GET#ing the first 2 bytes from each until the track value equals zero.

Other Program Notes

This program will work on the 8050, 8250, 9060, 9090, and the 4040 by removing the first "rem" in line 100. DOS variables in the 4040 are all in the same places but offset by 4. The single drive 2031 is totally different and will not respond properly to this program. VIC disks were not tested.

Accomodations are made for 21 simultaneous open files with a maximum size of 101 blocks each. The two integer arrays use up a total of about 8K. Change the dimensions appropriately for more or less files, and more or less maximum blocks, but they must be identical and there must be enough RAM to hold them along with your program and other variables. A quick approximation for RAM consumption of one integer array is the product of the dimensions times 2.

The D array is for Drive numbers. Preferably, all files

concerned would be on one diskette, but the routine will find files on either drive.

PRINT DS$ might be replaced by a GOSUB to your favorite error routine. Non–existent files will also be reported by lines 160 to 180.

Even though the data file has been CLOSEd (line 150), all DOS variables are still present in RAM until they're over-written by another OPEN. The sector, track, and drive number are used to "U1" the actual directory sector that holds the filename FI$. The track and sector co–ordinates of the first block in a file are stored in the 2nd and 3rd positions of a directory entry. Thus 1 is added to the offset before executing the "buffer–pointer" command.

Should the file you've OPENed be found in the last sector of the directory, the track, as with any other file, is set to zero. If Track = 0, T is set back to the directory track number (line 270), but this must be decided by the operator (eg. another INPUT) or, better yet, a subroutine designed to test what type of disk unit is connected.

Using This Technique

With this routine, a relative file system could be implemented with record sizes much larger than 254. Of course strings are limited to 255 so more than one would be necessary to store a record.

A subroutine for this would need the array index number (1st dimension) that contains the pointers to a particular file. This information should also be stored as pointers are collected. To position to a record, the program would "U1" the closest sector. The co–ordinates for this sector will be stored in the array at the element calculated by:

INT ((record number – 1)*(record size / 254))

The remainder of the above is used to position into the sector when the record size is not a multiple of 254. A counter must also be maintained so that the next block would be read when the 254th byte has been retrieved. A slight variation of Jim Butterfield's Universal String Thing (this issue) would be ideal.

Now that you have all this done, a columnar report generator is a simple task. Figures from several different files could be output side–by–side.

Although this approach is not immediately suited for expanding files, it could be included with a little extra code. However, changes to existing data can be performed in a fraction of the time it would take to open and close all of the files seperately.

On final note. . . Bill MacLean tells me that he may release a more complete version of this utility if there's enough demand. It would contain all necessary functions such as those discussed, plus some extras, (no doubt). Comments are welcome, please address them to me (Karl J. Hildon) at The Transactor.

```
100  rem d4040=4 : rem remove rem for use with 4040
110  z$=chr$(0) : dim t%(20, 100), s%(20, 100), d(20)
120  input"filename, file# ";f$, f : fi$=f$+",s,r"
130  open 2, 8, 2, "# " : open 15, 8, 15          : rem open direct and cmd channel
140  open 5, 8, 5, (fi$)                          : rem open and
150  print ds$ : close 5                          : rem close file 'fi$'
160  print#15, "m-r"chr$(142+d4);chr$(67)
170  get#15, a$ : e=asc(a$+z$)                    : rem error?
180  if e=255 then print"file not found" : print : goto 450
190  print#15, "m-r" chr$(146+d4);chr$(67)
200  get#15, a$ : s=asc(a$+z$)                    : rem sector in dir
210  print#15, "m-r"chr$(149+d4);chr$(67)
220  get#15, a$ : t=asc(a$+z$)                    : rem track in dir
230  print#15, "m-r" chr$(150+d4);chr$(67)
240  get#15, a$ : o=asc(a$+z$)                    : rem offset into sector
250  print#15, "m-r"chr$(144+d4);chr$(67)
260  get#15, a$ : d=asc(a$+z$) : d(f)=d           : rem drive num
270  if t=0 then t=39          : rem t=18 for 4040, 76 for 9060/90
280  print"entry dir track & sector ="t;","s
290  print"       offset into sector ="o
300  print"                   drive ="d
310  print#15, "u1:"2;d;t;s
320  print#15, "b-p"2;o+1                         : rem position into dir
330  n=0                                          : rem array index
340  get#2, a$                                    : rem get track
350  t=asc(a$+z$)
360  get#2, a$                                    : rem get sector
370  s=asc(a$+z$)
380  t%(f, n)=t : s%(f, n)=s                      : rem store in array
390  if t=0 then 450                              : rem track=0? yes, end
400  print"sec."n;"of file at"t,s
410  n=n+1
420  print#15, "u1:"2;d;t;s                       : rem read subsequent blks
430  print#15, "b-p:"2;0                          : rem pos'n to zero
440  goto 340
450  close 2 : close 15
```

Translation Arrays

How many times have you wanted to represent a number using a key on the keyboard? Never, eh. Well. . . next time you do, this tidy little piece of code will come in handy.

```
100  get a$ : if a$="" then 100
110  if a$="q" then x=1
120  if a$="w" then x=2
130  if a$="e" then x=4
140  if a$="r" then x=8
150  if a$="t" then x=16
160  if a$="y" then x=32
170  if a$="u" then x=64
180  if a$="i" then x=128
190  if a$="o" then x=256
200  if a$="p" then x=512
210  print x :  goto 100
```

But you say, "that's not very tidy at all" , and you're absolutely right. As a matter o' fact, it reeks. Here's a better way. . .

Given the nature of the possibilities for X (ie. all powers of 2), it becomes apparent that a FOR–NEXT loop could search through KY$, and then output 2 raised to the power of N, where N is the position of A$ within KY$. For example:

```
100  ky$="qwertyuiop"
110  get a$ : if a$="" then 110
120  for j=1 to len(ky$)
130  if a$ = mid$ (ky$, j, 1) then
     print 2↑(j–1) : goto 110
140  next : goto 110
```

Only the problem here is that as KY$ gets longer, the search becomes slower, and our output numbers may not be so orderly.

The best way is a translation array. It allows for expansion, it's fast, and can handle every key on the keyboard.

```
100  ky$="qwertyuiop"
105  dim out (255)
110  for j=1 to len(ky$)
120  out (asc (mid$ (ky$, j))) = 2↑(j–1)
130  next
140  get a$ : if a$="" then 140
150  print out (asc (a$)) : goto 140
```

Of course the above is merely a transposition of the first two. The technique shows its true strength in the following examples.

IEEE Modem Driver

Have you ever been in a bind for an ASCII modem driver but just didn't have one within reach? Converting PET ASCII (sometimes called "PETSCII") to real ASCII is not so tough using translation arrays. With this program you can be up and running in no time from any PET/CBM through an IEEE modem, and it's easily memorized.

```
100  gosub 200
110  get a$ : if a$<>"" then print#5, chr$(t(asc(a$)));
120  if peek(srq) and 128 then get#5, a$ : x9=peek(ieee) :
     print chr$(f(asc(a$+chr$(0))));
130  goto 110
200  dim t(255), f(255) : print chr$(14)
210  for j=32  to  64 : t(j)=j      : next
220  for j=65  to  90 : t(j)=j+32   : next
230  for j=91  to  95 : t(j)=j      : next
240  for j=192 to 218 : t(j)=j-128  : next
250  t(13)=13 : t(20)=8
260  rem add more functions here
270  for j=0 to 255 : if t(j) then f(t(j))=j : f(t(j)+128)=j : next
280  poke 1020, 0 : poke 59468, 14 : open 5, 5
290  srq=59427 : ieee=59426 : return
```

This modem driver is actually an adaptation of one by Jim Butterfield with a pinch of Paul Higginbottom flavouring. Notice that the keyboard/modem servicing is kept close to the start with the set-up called once as a subroutine. In a communications type environment, maximum speed is essential. When BASIC sees a backwards GOTO (as in line 130) it starts at the beginning of BASIC and searches forward to the destination line. By moving the setup routine out of the way, BASIC is relieved of looking through unnecessary line numbers. The time saved is not much, but might be just enough to save potentially lost characters while transmitting and/or receiving.

Two translation tables are used here: the T array for characters sent to the modem and the F array for characters received from the modem. Integer arrays could have been used but floating point arrays are faster.

Graphics are seldom used in communications. PRINT CHR$(14) sets Upper/Lower case mode.

Line 210 begins setting up the TO array. Characters 32 (space) through 64 (@) are the same in ASCII as in PETSCII, so a simple equate does it. In Upper/Lower case mode, PETSCII 65 to 90 displays lower case letters, and in true ASCII, lower case ranges from 97 to 122, so 32 is added to each (Line 220). The five characters from 91 to 95 are the same again for both. True ASCII upper case is the same as PETSCII lower case values, however, unlike dumb terminals, one must use the SHIFT key to obtain them. This is handled by line 240. Carriage return and Delete are away from the main stream of things so they're set up individually (line 250). POKE 1020, 0 disables IEEE timeout and the OPEN command is for the modem.

Getting back to our subject, it's lines 110 and 120 that take advantage of the arrays. Line 110 GETs characters from the keyboard. When it has one, the PETSCII value is used as an index into the TO array. For example, pressing DELete, which is CHR$(20), causes the contents of T(20) to be sent to the modem as a single byte value, in this case 8 which is ASCII for Rubout.

Line 120 GETs from the modem. SRQ signals a character pending by setting bit 7 of 59427. After the GET#, the SRQ flag is cleared by simply doing a read of the IEEE output buffer (this is built into the smarts of the IC). The true ASCII character received is used as an index into the FROM array. The character is converted to PETSCII and PRINTed using CHR$. The CHR$(0) is added to A$ as a precaution against the null string.

Keyboard Setup

Now let's take things one step further. Addressing one problem at a time is often the best way to solve a programming task. Here we'll look at a simple organ keyboard.

First, a table of frequencies must be established. This is done easily using a standard formula found in any book of music fundamentals. It starts with the highest desired note and then divides it down for subsequent descending notes. The results are store into the F (Frequency) array.

Next the keyboard is defined. On octave covers 12 notes, A to G plus the sharps. The 24 keys chosen to correspond to the 24 calculated frequencies were selected to resemble a dual-level piano keyboard (Lines 1000–1040).

The KY array is used to store ascending values from 1 to 24 in the element corresponding to the ASC value of the keys (Lines 2000–2040).

So now we have two levels of indirection. The ASC value of the key pressed is used to index the key array, and the value

there is used to index into the frequency array.

```
1000  dim f(24) : f1 = 7040
1010  for j = 24 to 1 step-1
1020  f1 = f1 / (2 ↑ (1/12))
1030  f(j) = f1
1040  next
2000  dim ky (255)
2010  ky$ = "zsxdcvgbhnjmr5t6yu8i9oØp"
2020  for j = 1 to len (ky$)
2030  ky (asc (mid$ (ky$,j,1))) = j
2040  next
3000  poke 59466, 15
3010  poke 59467, 16
3020  get a$ : if a$ = "" then 3020
3030  fr = (500000/4/f (ky (asc (a$)))) - 2
3040  poke 59464, fr      : rem print fr
3050  if peek (151) <> 255 then 3050
3060  poke 59464, 0
3070  goto 3020
```

Several modifications are available to this program. First of all, it uses only one waveform pattern. The ",15" in line 3000 is binary 00001111. This will produce a very even square wave. Try different values but 0 and 255 will produce nothing. Secondly, try substituting the single 4 in line 3030 to a 2 or an 8. The variable FR can range from 1 to 255. Take the REM out of line 3040 to see what it's doing. The range could be increased several ways. You might add more frequencies and keys or you can leave a couple of octaves gap between the lower and the higher when setting up the F array.

This program is extremely unsophisticated, so there's lots of room for improvement. However, the emphasis was on translation arrays. When we explore music synthesis, the techniques employed here will undoubtedly re-surface.

Filing It.

Jim Butterfield
Toronto, Ont.

Once you have learned how to input from the keyboard and output to the screen, it's easy to take the next step, and input or output using other devices.

The printer is handy, of course. But the super power comes from devices you can both read and write. That way, information can be stored now and brought back in later. You can store names, addresses, phone numbers. . .and read them in when you need them. This gives you two advantages: first, the information isn't lost when you turn the power off; and second, you have lots of storage space even if your computer memory is small.

The Golden Rule.

You'll find it easy to remember the golden rule of input and output: the information going out is almost exactly as you would see it on the screen. So if you asked to print out a value of 167, the following characters would be sent to the cassette, printer or disk: Space; 1; 6; 7; Return. That's almost exactly the same as would go to the screen; we wouldn't see the Return on the screen, but we'd see its effect since a new line would be started.

The opposite side of the golden rule concerns input. If the above value was written to a device, and later we rewind and ask to INPUT from that device, the program will received exactly the same information as if we typed on the keys: Space, 1, 6, 7, and Return. On the keyboard, RETURN signals that we are finished; and it means the same when the information comes from some other device.

Two special situations should be mentioned. You might have noticed that if we say PRINT 167 an extra character is delivered to the screen: behind the last digit, 7, there's a cursor-right. You may not notice it, since it doesn't print, but it's there. This extra character will not be sent to other devices. That's good because we don't need it; we save the space and no harm is done.

The other situation is another invisible character. Many versions of Basic send one more character after RETURN. Basic 4.0 does not normally send it, but most other Basics send a special character called a Linefeed. The Linefeed is a nice character for certain types of printers: it may be needed to move the paper up ready for printing the next line. But it's wasted in data storage, and might even give us a little trouble. More on this later.

Writing A File.

It's easy to write a file. All we need to do is: Open it, which tells the computer to get everything ready to go; Print the stuff; and then Close it, which tells the computer to wrap everything up.

Let's do it. If you have cassette tape, type:

```
OPEN 6, 1, 1, "DATAFILE"
```

or if you have disk, type:

```
OPEN 6, 8, 2, "0:DATAFILE,S,W"
```

. . .and in either case, your file number 6 is ready to go.

Now we can write a few things. Let's try some numbers:

```
PRINT#6, 3
PRINT#6, 123
PRINT#6, 3*45*6
```

And a few names:

```
PRINT#6, "HELLO"
PRINT#6, "MY NAME IS FRED"
```

Finally, we wrap up the file with:

```
CLOSE 6
```

A few notes. Did you notice that after we opened the file, the coding was the same no matter whether we were going to tape or disk? The OPEN statement sets everything up for us. This can make things very easy.

Note that we use one print statement for one item. Don't try punctuation: PRINT#6,3,123 would not work right — we will need that extra RETURN when we read back the data. It's also interesting to see that expressions are worked out before being printed, so that 3*45*6 will be placed on the file as value 42.

Now for that sneaky Linefeed. You don't really need to worry about this if you have 4.0 Basic or if you are using cassette tape, but it's good practice. Those PRINT# statements wrote the information we asked; then a Return, which we wanted; then a Linefeed, which we didn't want. We can get rid of the unwanted Linefeed by writing the Return ourselves — it codes as CHR$(13). So we might more correctly write:

```
PRINT#6, "HELLO"; CHR$(13) ;
```

. . .and don't forget both semicolons.

Reading It Back.

This is just as easy, except that we need to write these statements as a program. INPUT and INPUT# won't work as direct statements typed on the screen. So we code:

```
100 OPEN 4, 0, 0, "DATAFILE"
```

or, for disk:

```
100 OPEN 4, 8, 3, "DATAFILE"
```

And continue with:

```
110 INPUT#4, A$
120 PRINT A$
130 IF ST = 0 GOTO 1100
140 CLOSE 4
```

What's ST doing? That's the Status word. If it's zero, we are reading our file normally. If it's non-zero there is something going on — usually we are at the end of the file (ST will equal 64 in this case).

Your data should come back very nicely just as you wrote it.

Conclusion.

It's not hard to write and read files. We'll pick up a few fine points next time around.

SuperPET Terminal Program

John Stoveken
Milton, Ont.

If you have a SuperPET and an RS232 modem, then you probably have a terminal program to make them go. But unless you wrote the program, you may be wondering how the SuperPET serial port works. This program will show you just how easy it is.

The SuperPET RS232 port is controlled entirely by one chip; the 6551 ACIA (Asynchronous Communications Interface Adapter). The 6551 has 4 internal registers; dta, status, the command register, and the control register. Incoming/outgoing characters are routed through the Data Register. The Status Register tells us if a character has arrived, if the last one was sent, and other things like transmission errors. The Command Register controls various transmit/receive functions such as parity mode, full/half duplex, etc. Baud rate, word length, number of stop bits, and clock source are controlled by the Control Register. Addresses for the 6551 registers are from $EFF0 to $EFF3.

Only the Control and Command Registers need to be set up before using the port. Various functions are invoked by setting bits within the registers. The program sets the Control Reg. for 300 baud (baud=bits per second), a word length of 7 with 1 stop bit, and all controlled by the 6551 internal clock. The Command Reg. is set for even parity and no interrupts. Some other configurations are shown in the listing, but for more details see the tables that follow.

Once you've established the configuration of the ACIA, the Data and Status registers do all the work. The Data Reg. is used for both sending and receiving characters. The 6551 "knows" the difference between characters intended for output and those that must be treated as input. Two bits in the Status Register flag these conditions. When a character comes in from the RS232 line, bit 3 of the Status Reg. is set to 1 (lines 220–230). If you're sending a character by POKing it to the Data Register, bit 4 fo the Status Reg. must be set to 1 or else the last character has not been sent (lines 340 & 350). The entire communications section in this program lies between 200 and 390. It's easy to follow. . . start by imagining an inactive RS232 line and an inactive keyboard.

The Status Register also flags other potential conditions such as errrors in transmission. But unless you go to a machine language program, you won't have time in BASIC to do anything with them. BASIC will permit about 300 baud maximum. Anything higher and you'll start dropping characters.

Editor's Note

You'll notice that John is using translation arrays to convert Pet ASCII to true ASCII and vice versa. For more information on this powerful and versatile technique, see the article in this issue entitled "Translation Arrays" .

```
10 rem **** simple basic terminal program for superpet ****
20 rs = 14*4096 + 15*256 + 15*16:rem superpet 6551 address
30 cn = 6 + 16 + 32 + 0 : rem 300 baud + clk + 7 bit word + 1 stop bit
31 rem other configs:6 + 16 + 32 + 128 = above with 2 stop bits
32 rem                8 + 16 + 32 + 0   = 1200 baud, good luck in basic
33 rem                6 + 16 + 0  + 0   = 8 bit word
40 cm = 96 + 11:rem even parity and no interrupts
41 rem  32 + 11 = odd parity,0 + 11 = no parity
50 poke rs + 2 , cm
60 poke rs + 3 , cn
70 d$=chr$(20)
80 gosub 1000          :rem set up translation arrays
90 print"[S]";
98 rem                      *******        get character in        *******
99 :
200 a = peek(rs+1)                         :rem status register
210 if (a and 8) = 0 then 300              :rem no character received
220 cr = peek(rs)                          :rem receive character
225 cr = ap(cr)                            :rem ascii to pet conversion
230 print chr$(cr);
297 :
298 rem                      *******        send character out      *******
299 :
300 get a$ : if a$="" then 200             :rem get keyboard character
310 ch = asc(a$)
320 if ch=18 then ct=1 : goto 300          :rem next character control
330 if ct=1 then ct=0 : ch=ch and 63
340 a = peek(rs+1)
350 if a and 16 = 0 then 340               :rem last character not sent
360 print chr$(ch);
370 ch = pa(ch)                            :rem pet to ascii
380 poke rs, ch                            :rem send character out
390 goto 200
997 :
998 :rem          ascii to pet to ascii table conversion
999 :
1000 dim pa(255), ap(255)
1010 for i=0 to 64 : pa(i)=i : ap(i)=i : next :rem numbers and stuff
1020 for i=65 to 90
1030 pa(i) = i + 32
1040 ap(i) = i + 128
1050 next
1060 for i=91 to 96 : pa(i)=i : ap(i)=i : next : ap(96)=44
1070 for i=97 to 127
1090 ap(i)=i-32
1100 next
1110 for i=193 to 218
1120 pa(i)=i-128
1130 next
1140 ap(127)=20 : pa(20)=127               :rem deletes
1150 ap(8)=157 : pa(157)=8                 :rem backspace
1160 pa(7)=7 : ap(7)=7                     :rem bell
1170 ap(12)=147 : pa(147)=12               :rem clr screen/form feed
1180 ap(10)=10 : pa(10)=10                 :rem lf
1190 pa(13)=13 : ap(13)=13 : pa(141)=13:rem cr (permits shift lock)
1200 return
```

APL And The SuperPET
Serial Port

Eike Kaiser
Toronto, Ont.

After working with computers for only four years, the real surprise in this relationship is probably the infrequency rather than the frequency with which major problems arise. I won't claim that this is because almost all my work is in APL, but it probably does contribute. One such problem that I've finally cleared up involved the use of the serial port (RS–232C) on my SuperPET. It took a lot of help from a lot of people, but my special thanks go to Peter Velocci of Commodore Canada. He took the time and showed the interest to set up the contacts that finally shed light on one more hidden secret of the SuperPET.

Documentation of the SuperPET's serial port has only been released in bits and pieces. At this point I'm still not convinced that enough pieces have come out to let the average user pull them all together, even if he happens to have access to all the available literature. I'll try to correct this here, at least for any APL users in the audience. Don't stop though, if you are not an APL freak like me. Much of this is not tied to any one language, other parts may at least be a guiding milestone!

The first problem in using the serial port is to get some information about its "pinouts": the location and operation of each of the 25 electrical connections that make up this communications channel. Some of this has recently become available, but none that I found was either explicit or complete. Here is how the standard information goes:

RS–232C has the following 25 pins, in two columns

1	earth ground	14	sec. transmitted data
2	tansmitted data (TXD)	15	transmit clock
3	received data (RXD)	16	secondary received data
4	request to send (RTS)	17	receiver clock
5	clear to send (CTS)	18	. . .unassigned. . .
6	data set ready (DSR)	19	sec. request to send
7	logic ground	20	data terminal ready (DTR)
8	carrier detect (DCD)	21	signal quality detect
9	. . .reserved. . .	22	ring detect
10	. . .reserved. . .	23	data rate select
11	. . .unassigned. . .	24	transmit clock
12	sec. carrier detect	25	. . .unassigned. . .
13	sec. clear to send		

Normally, only pins 1–8 and pin 20 are used. Diablo and SuperPET use only these pins but may have conflict on pins 2 & 3, or pins 4 & 5.

REF: Byte, May 1982, p. 212 et al
Commodore Mag, Feb 82, p.58, Apr/May 82 p.87
Diablo documentation, p.2–10
Torpet, April 1982, p.23
Computers and Programming, Jul/Aug 1981, p.31
The Transactor, Volume 3, Issue #6, p.6

The pin allocations shown above reflect the industry stand-

ard which, so far as I have seen, Commodore have adhered to. As noted, only pins 1–8 and pin 20 are used by the SuperPET. Among other things, this limits users to only one serial device at a time being attached to the machine. If you dreamed of a plotter and printer, one of them better be on the IEEE port.

The note on a possible conflict on pins 2 and 3 turned out to be a fact. I doubted this for a long time. The connection inside the SuperPET is the female side of the RS–232 plug. Without wishing to sexist, I reasoned that the "transmitted" line on a male plug really ought to be the "received" line on a female plug. After all, none of the documentation referenced above said "transmitted to" or "transmitted from". My guess seemed a logical compromise, right???

Not so. It turns out that male or female doesn't matter (at least not in computers). Pin 2 sends data from whatever device that pin is attached to, regardless of its sex! Count one more for equality.

The moral of all this is that pins 2 and 3 MUST be reversed in any connection between a SuperPET and a serial device that adheres to the industry standard for the RS–232C ports. That is a generalization gleaned from experience, but it seems even more sensible than my male/female theory.

The possible conflict on pins 4 and 5 relates to a "handshake" procedure, where the SuperPET confirms that the printer is ready for input. Pins 6 and 8 are used to test that the printer is in fact there. If your printer doesn't actually use these pins then it's time to play electrician. Without removing any existing connections, somewhere in either machine, or in the cable joining them, you must make a connection between pins 4 and 5, and/or pins 6, 8, and 20. These fool the SuperPET into thinking that the printer is sending all the right acknowledgements of its existence. For details, Waterloo has a technical letter with a bit more information.

So much for the hardware. Now just plug it all up and watch APL fly back and forth between machines — right? Wrong!

The SuperPET has its own special ASCII representations, which differ from the standard that most manufacturers adhere to. This is more of a nuisance than a problem, since Waterloo have provided some facility, however meagre, for converting this "internal representation" into an "external representation" that devices like ASCII printers can use. Conveniently, these are provide as the system functions "⎕XR" and "⎕IR". The former converts SuperPET jargon into something that the rest of the world can understand, while the latter turns incoming code into the SuperPET's talk.

This difference doesn't seem to affect other Waterloo languages, perhaps because they use the standard typewriter character set, unlike the special characters used by APL. Waterloo may also have found a less cumbersome way of handling the conversion in the other languages, and are moving slowly in this direction in release 1.1 of APL. Note though, that in release 1.0 these conversions do not work on matrices. This is the improvement that release 1.1 will bring. For now we must convert our text line by line.

Note that I said "text". Waterloo didn't provide for sending anything other than characters out the serial port, at least in APL. Numbers and function listings don't qualify. Both must first be converted to character representations of themselves. This again is quite contrary to the basic philosophy of APL, which was to let users be users, and oblivious of computer technicalities.

Fortunately these are not difficult tasks. "Thorn" (⍕) is the APL operator that converts numbers into their character representation. "⍕5 5*pi*25" gives a printable 5 by 5 matrix of all the numbers from 1 through 25. Functions, the APL equivalent of user defined programs, are converted by the statement "⎕CR 'FN' ", where FN represents the function's name, which must be enclosed in single quotes. This has a hidden problem though. ⎕CR strips the function of its line numbers, making the printed listing very different from users, especially new users, come to expect from their experience with listing functions on the screen. Thus, yet another step is forced on us by Waterloo — re–inserting line numbers in a way that mirrors what APL would normally give us. The program "NUMBER" is included below to meet this need.

Here then are our three functions. "PRINT" is the main routine, and fairly thorough notes on its operation follow the listings. "ASK" is a utility function that is called by PRINT, its purpose being to modify the way that user input is handled at the screen and to interrupt program execution until that input is received. This is used to permit changing paper if you are not using continuous forms. An alternative statement allowing for automatic advance of continuous forms is provided in the discussion below. "NUMBER",finally, provides for function line numbering as described above.

Do note that the problems we are fixing here are not APL problems. They are little inconsistencies that Waterloo gave us in an otherwise very acceptable APL implementation. Hopefully future releases, beyond 1.1, will continue to work towards the APL goal of encouraging a very casual user relationship with the computer.

Discussion of Function "PRINT"

Note: See end of article for program listings.

APL programs alway start with line zero, which defines the syntax of the function, along with any local variables that it will create. Their names, separated by semi–colons, list the temporary work variables that will no longer be needed when execution terminates. Having them here tells APL to throw them out for you. Any variables created by the executing function but not listed in this way remain available for possible later use.

"PRINT" expects a right argument of X, which is the data we want printed. "X" can be a variable name, it can be "⍕DATA" (the character representation of numbers) or it can be "□CR 'FN' ", as discussed above. In fact, APL even lets us string together more complicated arguments to functions. Remember the problem of line numbers not appearing in the □CR of a function? To fix this, the "NUMBER" function above can be used as in this statement:

PRINT NUMBER □CR 'FN'

In this case "NUMBER" is just another function which does its job before PRINT. The right argument in the header permits this type of syntax.

Line one of an APL function is often reserved for a brief statement of what the function does or how. This is strictly a matter of style. I suggest you get into the habit, and a future article will show how you can take great advantage of this feature. Unlike BASIC, APL lets you have numerous programs and sets of data available simultaneously, mixing and matching them as in our "PRINT NUMBER" example. This saves a great deal of code, because you can re–use utilities rather than re–coding them into every application that needs them. With such a mixture of functions, that one line of documentation can protect against confusion over similar names, and help you or others find their way around.

Line 2 imposes a condition of our own on the data to be printed. This can be removed, if you prefer, along with its error report on line 13. Its purpose is to prevent printing anything except two–dimensional matrices. The reason for this is that it is much easier to re–structure the data to a standard format than to generalize the program for n–dimensional output.

Line 3 sets up the ASCII printer as the output port. Even though "serial" is a pre–defined file to the SuperPET, it is necessary to 'CREATE' it for use rather than just 'TIE' it. To me this represents another anomally in the Waterloo implementation. . . but again too small to beef about. As in BASIC, files are accessed by number. An APL system variable, □NUMS, keeps a list of all of the currently active file 'TIE' numbers, and the "⌈/□NUMS" returns the largest element in this set. This illustrates the "monadic" use of "⌈", or ceiling, combined with reduction, "/", to select only the largest element from a set of data. Our statement also uses "⌈" in its 'dyadic' sense, maximum, which selects the larger of this primitive's left and right arguments. In this case we have 0 on the left, and the result of "⌈/□NUMS" on the right. Thus if any files are already TIEd then they will have a TIE number larger than zero, and that will be the maximum of the two values. If no files are TIEd, then 0 will be the larger of the two values. In either case, we add one to the resulting value, to create a new and currently unused TIE number. This is stored in the variable "TIE", with which we then"□CREATE" the link to the serial port.

At this point we should test for the success of this operation. Release 1.0 would allow this, but under release 1.1 of APL there will be no return code from the "□CREATE" operation, and therefore nothing to test. (This is based on preliminary release 1.1 information, and remains to be proven when this release becomes available) For now we will leave the function in the form most likely to outlive the imminent transition to release 1.1.

Line 4 now defines the number of rows of data (in X) that will be printed. This is used later to test for completion of the printing to be done. It wasn't necessary to use a variable to provide this control, but then we would have had to interrogate the system for the size of X after each line was printed. In some APL systems this can waste time and computer power, especially when large numbers of iterations are involved.

Line 5 initializes a counter to zero. This will count the iterations through the printing loop, and hence the number of lines printed. It is immediately incremented on line 6, which starts the loop to sequentially print each line of data.

Line 7 does the actual printing, but it also includes a test for the success of this operation. Note that each line of data will have two characters (carriage return and linefeed, from □TC) catenated to it's end before being turned to its external representation and then □PUT to the file "TIE". If this operation is not successful it returns an error message which we store in "Q". Next we test if 0 is not equal to the shape of Q (ie. that Q is not empty). If this is true (Q is not empty) then Q must contain an error message and we branch to the line "OUTERR" (line 15) where the user is advised of the failure and it's cause.

Line 8 now tests whether we have printed an even multiple of 60 lines. It uses the concept of residuals, and the assump-

tion that most paper forms will accommodate 60 lines of print. You can change this value to suit your own needs. If the residual of "I" relative to this base value is not zero, then the program branches past line 9.

If this residual is zero (lines 60, 120, 180,. . .) then line 9 invokes the sub–function "ASK". This prints to the screen the message that is given as it's right argument: here, "PAPER OUT. . .". This sub–function then waits for a response from the operator, allowing whatever time is necessary to permit changing paper. If you are using continuous forms with 66 lines per page, replace this line with:

(□XR 5*p*□TC[3])□PUT TIE

This will space the paper up by 5 lines, getting you well past the perforations in the forms. For shorter forms you can modify the value 60 in line 8, and even squeeze down the vertical spaces in this revision to line 9.

Line 10 test whether "I" is less than "LINES" — have all available lines been printed? If not, it branches back to the top of LOOP1 for the next line of data. If so, it proceeds to line 11, the equivalent of a "CLOSE" statement in BASIC. Lines 12 and 14 branch out of the program to avoid printing error messages on 13 and 15, which were provided for conditions that might arise on lines 2 and 7.

The utility function "ASK" will not be explained here. It is well worth playing with this one though, since with small modifications it provides a wide range of control over user input. For example, change it to expect a left argument representing acceptable responses, then test for those responses and branch back to line 2 if the user input doesn't match the allowed inputs. This is the basis of an almost unbeatable trap to safeguard your programs against all sorts of evils!

The function "NUMBER" will also not be explained here. At this point we don't know what audience the new Transactor will gather to itself, and the last thing we want is to turn off any of that audience with long monologues that don't meet their needs.

Our purpose was to unlock some of the secrets of the SuperPET's serial port, particularly for APL users who seem to have had an unfair share of undocumented problems to face. In the process we have tried to illustrate some concepts underlying the APL language. Other secrets are left for you to face, such as the parity, duplex, and linefeed settings on equipment that is available to you. Various suppliers and retailers will no doubt be glad to help with further questions, including Waterloo. Patience and experimentation on your part will be invaluable though, since none of these sources will have had the experience with every gadget you might want to plug in to the RS–232C port on you machine.

Let us know how you feel about this article. Did it help with the serial port? With introductory APL examples? With example of APL style?

Did it confuse?. . . bore?. . . enlighten? Was it timely or out–of–date, too simple or too advanced? Your interests are our interests, because our magazine is your magazine.

```
      ∇PRINT[□]∇
[0] PRINT X ;LINES;I;Q;TIE
[1]⍝ PRINTS X TO PRINTER VIA SERIAL PORT; NOT LIMITED TO SCREEN □PW
[2]     →(2≠ρρX)/IPTERR
[3]       'SERIAL'□CREATE TIE←1+0⌈⌈/□NUMS
[4]       LINES←1↑ρX
[5]       I←0
[6]LOOP1:I←I+1
[7]     →(0≠ρQ←(□XR X[I;],□TC[7 3])□PUT TIE)/OUTERR
[8]     →(0≠60|I)/TEST
[9]       Q←ASK 'PAPER OUT:  ¨RETURN¨ TO CONTINUE.'
[10]TEST:→(I<LINES)/LOOP1
[11]     □UNTIE TIE
[12]   →0
[13]IPTERR:'INPUT MUST BE 2 DIMENSIONAL:  RE-STRUCTURE AND RE-SUBMIT.'
[14]   →0
[15]OUTERR:'INTERRUPT:  OUTPUT FAILED, ',Q
     ∇


      ∇ASK[□]∇

[0] R←ASK X
[1]⍝ ALLOWS RESPONSE ON SAME LINE AS PROMPT X
[2]  ⍞←X,' -- '
[3]  R←(4+ρX)↓⍞
     ∇


      ∇NUMBER[□]∇

[0] Z←NUMBER TXT;R
[1]   R←1↑ρTXT
[2]   R←0 ¯1↓⍒(R,1)ρ¯1+⍳R
[3]   Z←('[',R,']'),TXT
     ∇
```

Tiny–Aid For VIC–20

David A. Hook
Barrie, Ont.

Introduction

Since the early days of the PET, various enhancements for BASIC have been available—Toolkit and Power are two commercial examples. Bill Seiler, then of Commodore, produced the first public–domain version, called BASIC–Aid.

Many updates, corrections and improvements have been made over the past couple of years. The PET/CBM program has ballooned to a 4K package for almost every flavour of equipment configuration.

As has been customary in the Commodore community, Mr. Jim Butterfield developed a version of the BASIC–Aid. He called this TINYAID2 (or TINYAID4, for Basic 4.0). This offered the six most useful commands from the full–fledged program.

Following is my modification of that work, designed to provide VIC users with the same benefits. After using this for a while, I think you will find the added commands nearly indispensible.

Features

VIC Tiny Aid is a machine language program which consumes about 1200 bytes of your RAM memory. After you have loaded the program, type 'RUN' and hit 'RETURN'. The program repacks itself into high memory. The appropriate pointers are set so that BASIC will not clobber it. VIC Tiny Aid is now alive.

Once activated, five commands become attached to BASIC. They will function only in "direct" mode, i.e. don't include them in a program.

(1) NUMBER 1000,5 'RETURN'
NUMBER 100,10

Renumbers a BASIC program with a given starting line number and given increment between line numbers. The maximum increment is 255.

All references after GOTO, THEN, GOSUB and RUN are automatically corrected. A display of these lines is presented on the screen as it works. If a GOTO refers to a non–existent line number, then it is changed to 65535. This is an illegal line number, and must be corrected before the BASIC program is used.

(2) DELETE 100–200 'RETURN'
DELETE– 1500
DELETE 5199 –

Deletes a range of lines from a BASIC program. Uses the same syntax as the LIST command, so any line–range may be specified for removal. DELETE with no range will perform like a NEW command, so be careful.

(3) FIND /PRINT/ 'RETURN'
FIND /A$/, 150–670
FIND "PRINT", 2000–

Will locate any occurences of the characters between the "/" marks. Almost any character may mark the start/end of the string to be found, so long as both are the same. The first example will find all the PRINT instructions in the program.

If you are looking for a string of text which contains a BASIC keyword, you must use the quote characters as markers. This will prevent the search string from being "tokenized".

If a limited line–range is desired, use the same syntax as for LIST. Note that a comma (",") must separate the line–range from the end marker.

All lines containing the string are printed to the screen. If a line has more than one of them, each occurence will cause a repetiton of that line.

(4) CHANGE –PRINT–PRINT#4,– 'RETURN'
CHANGE /ABC/XYZ/, 6000–
CHANGE /DS$/D1$/, –5000

Using the same syntax as FIND, you may change any string to any other string in a BASIC program. This command is very powerful, and was not part of the early versions of Basic–Aid or Toolkit.

As before, you may indicate a line–range. As the changes are made, the revised lines are displayed on the screen.

Watch out for the difference between BASIC keywords and strings of text within quotes. You may use the quote characters to differentiate, as with FIND.

(5) KILL 'RETURN'

This command disables VIC Tiny Aid and its associated commands. A syntax error will be the result if any of the above commands are now tried.

Since the routine is safe from interference from BASIC, you may leave it active for as long as your machine stays on. It is possible that VIC Tiny Aid may interfere with other programs that modify BASIC's internal 'CHRGOT' routine. The KILL command allows you to avoid this conflict.

Procedure

The VIC contains no internal machine language monitor, which is really the only practical way to enter this program. So follow one of the three methods below to perform the task.

(1) Borrow an Upgrade (2.0) or Basic 4.0 PET/CBM, with its internal ML monitor. This will be the easiest method to work with the program included.

(2) Use your VIC–20, but you must have a machine language monitor:

–Jim Butterfield's TINYMON FOR VIC (Compute#20, January 1982). –my adaption of SUPERMON FOR VIC (The Transactor, Volume 3, Issue #5). –VICMON cartridge from Commodore.

(3) The easy way (?). Send $3, a blank cassette or 1540/2031/4040 diskette in a stamped, self–addressed mailer to me at:

58 Steel Street
BARRIE, Ontario, CANADA
L4M 2E9

Be sure its packaged securely. Diskettes will be returned in DOS 2.0 format. Only 2040 (DOS 1.0) owners need take extra care. (The programs need to be copied to a DOS 1.0 formatted disk. Don't SAVE or otherwise WRITE to the disk you get).

If you are using a VIC, and have a 3K RAM or SUPEREXPANDER cartridge, plug this in. It will be somewhat easier to follow, since programs are then "PET–compatible" without further juggling. However don't use the 8K or 16K expansion for this job.

If you are familiar with the operation of the ML monitor, please skip ahead to the specifics below.

You are about to type in about 2500 characters worth of "hexadecimal" numbers. In addition to the digits from zero to nine, the alphabetic characters from A–F represent numbers from ten to fifteen. These characters, and three instructions, will be all that are used to enter our program. You don't have to understand the process—just type in the

characters exactly. It's not very exciting, but don't be too intimidated by the "funny" display.

Enter the machine language monitor program :

TINYMON/SUPERMON FOR VIC — LOAD and RUN the program.
PET/CBM — Type "SYS1024" and hit "RETURN" .
VICMON Cartridge — "SYS 6*4096" or "SYS 10*4096" (depending on version you have), then type "RETURN" .

NOTE: If you are working on the unexpanded VIC you will need to follow the alternate instructions in parentheses.

The cursor will be flashing next to a period character ("."). Type the entry starting at the current cursor position:

.M 0580 05C0 "RETURN" (.M 1180 11C0)

Several lines should appear on the screen, much like the "memory–dump" which accompanies this article. A four–"digit" quantity called an "address" leads off a line, and either eight or five columns of two– "digit" values appear alongside.

Look at the tables of values in the article. They show eight rows of these addresses. Note that the first "block" has the address "0580" , which matches the first address just above. The first row of the next table shows "05C0", which is the second (or ending) address just above.

Your mission is to type in the matching values from the article, in place of the two–"digit" values you see on the screen.

Remember to hit "RETURN" at the end of each screen line, or the changes won't be made.

Double–check the values you've typed. It's not easy to find an error later on.

Look at the next block of values. Type in the start/end addresses to display:

.M 05C0 0600 "RETURN" (.M 11C0 1200)

Type in the values required and go on with the rest of the blocks.

You will use addresses ranging from:

05xx–06xx–07xx–08xx–09xx–0Axx

as shown in the tables. The "x" characters stand for the other two "digits" of the address in the leftmost column.

If you are working on the unexpanded VIC, the sequence of addresses is:

11xx–12xx–13xx–14xx–15xx–16xx

You will have to type these pairs of characters in place of the leading two shown just above.

With that task complete, we are ready to preserve this work on tape. So type:

.S "VIC AID.ML" ,01,0580,0AB6 "RETURN"

(or .S "VIC AID.ML" ,01,1180,16B6 "RETURN")

Mount a blank tape, and follow the instructions. Save a second copy, for safety.

Exit the ML monitor, with:

.X "RETURN"

VERIFY the program normally before going any further.

Now comes the easy part. Type "NEW", then the BASIC listing. Enter this exactly, without including any extra text. Save this as "VIC AID.BAS" and VERIFY it.

Finally, LOAD "VIC AID.ML" and SAVE "VIC AID.REL" on another blank tape. Both the BASIC part and the machine language part have been SAVEd together.

Check–Out

We are going to check out the machine language using a "checksum" method. Type in "NEW" before proceeding. Now enter the following program:

```
10  i=0 : rem (i=3072 for unexpanded VIC)
20  t=0 : for j= 1408+i to 2741+i
30  t=t+peek(j)
40  next j
50  print t
```

After a few seconds, if the value 161705 appears, you've likely got it perfect. Go to the next section.

If not, there's at least one incorrect entry. Change the two

values in Line 20, using the table below. Re–RUN the program and compare against the value in the third column.

Repeat the process for each row, noting any that don't match. Each row corresponds to two "blocks" from the last section. You will have to re–enter the ML monitor to re–check those sections that differ. Re–SAVE the ML part!!!

Block#	Value 1	Value 2	Checksum
1– 2	1408	1535	15201
3– 4	1536	1663	17221
5– 6	1664	1791	15925
7– 8	1792	1919	15117
9–10	1920	2047	15565
11–12	2048	2175	14141
13–14	2176	2303	15840
15–16	2304	2431	16276
17–18	2432	2559	15152
19–20	2560	2687	15194
21	2688	2741	6073

Operation

The final acid test. ReLOAD the program from tape and RUN it. The screen will clear and a brief summary of the added commands will be displayed. The cursor should return almost instantly, under the "READY." message.

If the cursor does not come back, there is something still amiss. All the values appearing in the article were produced from a working copy of the program (Honest!). You still have option (3) from the Procedure section available. If you do send a tape/disk now, include your non–functioning version. I can then do a compare, to see where the error(s) were.

This has been a massive exercise, and mistakes can easily creep in. Your comments are welcome.

```
 1 print "■S■r■ vic tiny aid"
 2 print "■q■ adapted for vic by:"
 3 print " david a. hook"
 4 print "■q■ from 'tiny aid' by:"
 5 print " jim butterfield"
 6 print "■q■ and 'basic aid' by:"
 7 print "bill seiler"
 8 print "■q■■r■sample commands:"
 9 print "■q■change /?/print#4,/"
10 print "find .gosub., 200–"
11 print "delete 130–625"
12 print "number 100,5"
13 print "kill      (vic aid)"
14 sys (peek (43)+peek (44)*256+383)
```

```
.: 0580 a5 2d 85 22 a5 2e 85 23
.: 0588 a5 37 85 24 a5 38 85 25
.: 0590 a0 00 a5 22 d0 02 c6 23
.: 0598 c6 22 b1 22 d0 3c a5 22
.: 05a0 d0 02 c6 23 c6 22 b1 22
.: 05a8 f0 21 85 26 a5 22 d0 02
.: 05b0 c6 23 c6 22 b1 22 18 65
.: 05b8 24 aa a5 26 65 25 48 a5

.: 05c0 37 d0 02 c6 38 c6 37 68
.: 05c8 91 37 8a 48 a5 37 d0 02
.: 05d0 c6 38 c6 37 68 91 37 18
.: 05d8 90 b6 c9 df d0 ed a5 37
.: 05e0 85 33 a5 38 85 34 6c 37
.: 05e8 00 aa aa aa aa aa aa aa
.: 05f0 aa aa aa aa aa aa aa aa
.: 05f8 aa aa aa aa aa aa aa aa

.: 0600 df ad fe ff 00 85 37 ad
.: 0608 ff ff 00 85 38 a9 4c 85
.: 0610 7c ad d9 fb 00 85 7d ad
.: 0618 da fb 00 85 7e 4c 8f fc
.: 0620 00 f0 03 4c 08 cf a9 c9
.: 0628 85 7c a9 3a 85 7d a9 b0
.: 0630 85 7e 60 db fb 00 85 8b
.: 0638 86 97 ba bd 01 01 c9 8c

.: 0640 f0 10 d0 02 a4 8c a6 97
.: 0648 a5 8b c9 3a b0 03 4c 80
.: 0650 00 00 60 bd 02 01 c9 c4
.: 0658 d0 ed a5 8b 10 02 e6 7a
.: 0660 84 8c a2 00 00 86 a5 ca
.: 0668 e8 a4 7a b9 00 00 02 38
.: 0670 fd d9 ff 00 f0 13 c9 80
.: 0678 f0 13 e6 a5 e8 bd d8 ff

.: 0680 00 10 fa bd d9 ff 00 d0
.: 0688 e4 f0 bf e8 c8 d0 e0 84
.: 0690 7a a5 a5 0a aa bd f5 ff
.: 0698 00 48 bd f4 ff 00 48 20
.: 06a0 e9 fb 00 4c 73 00 00 20
.: 06a8 b2 fd 00 a5 5f a6 60 85
.: 06b0 24 86 25 20 13 c6 a5 5f
.: 06b8 a6 60 90 0a a0 01 b1 5f

.: 06c0 f0 04 aa 88 b1 5f 85 7a
.: 06c8 86 7b a5 24 38 e5 7a aa
.: 06d0 a5 25 e5 7b a8 b0 1e 8a
.: 06d8 18 65 2d 85 2d 98 65 2e
.: 06e0 85 2e a0 00 00 b1 7a 91
.: 06e8 24 c8 d0 f9 e6 7b e6 25
.: 06f0 a5 2e c5 25 b0 ef 20 33
.: 06f8 c5 a5 22 a6 23 18 69 02

.: 0700 85 2d 90 01 e8 86 2e 20
.: 0708 59 c6 4c 67 e4 20 7c c5
.: 0710 20 73 00 00 85 8b a2 00
.: 0718 00 86 49 20 8c fd 00 a5
.: 0720 a5 c9 00 00 d0 07 a2 02
.: 0728 86 49 20 8c fd 00 20 73
.: 0730 00 00 f0 03 20 fd ce 20
.: 0738 b2 fd 00 a5 5f a6 60 85

.: 0740 7a 86 7b 20 d7 ca d0 0b
.: 0748 c8 98 18 65 7a 85 7a 90
.: 0750 02 e6 7b 20 ca ff 00 f0
.: 0758 05 20 dc fd 00 b0 03 4c
.: 0760 8f fc 00 84 55 e6 55 a4
.: 0768 55 a6 31 a5 32 85 8b b1
.: 0770 7a f0 d8 dd 00 00 02 d0
.: 0778 ed e8 c8 c6 8b d0 f1 88

.: 0780 84 0b 84 97 a5 49 f0 5b
.: 0788 20 f0 fd 00 a5 34 38 e5
.: 0790 32 85 a7 f0 28 c8 f0 ca
.: 0798 b1 7a d0 f9 18 98 65 a7
.: 07a0 c9 02 90 40 c9 4b b0 3c
.: 07a8 a5 a7 10 02 c6 8b 18 65
.: 07b0 0b 85 97 b0 05 20 24 fe
.: 07b8 00 f0 03 20 0c fe 00 a5

.: 07c0 97 38 e5 34 a8 c8 a5 34
.: 07c8 f0 0f 85 8c a6 33 bd 00
.: 07d0 00 02 91 7a e8 c8 c6 8c
.: 07d8 d0 f5 18 a5 2d 65 a7 85
.: 07e0 2d a5 2e 65 8b 85 2e a5
.: 07e8 7a a6 7b 85 5f 86 60 a6
.: 07f0 43 a5 44 20 3d fe 00 20
.: 07f8 e1 ff a9 00 00 85 c6 a4

.: 0800 97 4c f2 fc 00 a4 7a c8
.: 0808 94 31 a9 00 00 95 32 b9
.: 0810 00 00 02 f0 15 c5 8b f0
.: 0818 05 f6 32 c8 d0 f2 84 7a
.: 0820 60 c9 ab f0 04 c9 2d d0
.: 0828 01 60 4c 08 cf 90 05 f0
.: 0830 03 20 a6 fd 00 20 6b c9
.: 0838 20 13 c6 20 79 00 00 f0

.: 0840 0b 20 a6 fd 00 20 73 00
.: 0848 00 20 6b c9 d0 e0 a5 14
.: 0850 05 15 d0 06 a9 ff 85 14
.: 0858 85 15 60 20 ca ff 00 85
.: 0860 43 20 ca ff 00 85 44 38
.: 0868 a5 14 e5 43 a5 15 e5 44
.: 0870 60 a5 7a 85 22 a5 7b 85
.: 0878 23 a5 2d 85 24 a5 2e 85

.: 0880 25 60 a5 22 c5 24 d0 04
.: 0888 a5 23 c5 25 60 a4 0b c8
.: 0890 b1 22 a4 97 c8 91 22 20
.: 0898 01 fe 00 d0 01 60 e6 22
.: 08a0 d0 ec e6 23 d0 e8 a4 0b
.: 08a8 b1 24 a4 97 91 24 20 01
.: 08b0 fe 00 d0 01 60 a5 24 d0
.: 08b8 02 c6 25 c6 24 4c 24 fe

.: 08c0 00 a0 00 00 84 a5 84 0f
.: 08c8 20 cd dd a9 20 a4 a5 29
.: 08d0 7f 20 d2 ff c9 22 d0 06
.: 08d8 a5 0f 49 ff 85 0f c8 b1
.: 08e0 5f f0 19 10 ec c9 ff f0
.: 08e8 e8 24 0f 30 e4 84 a5 20
.: 08f0 7c fe 00 c8 b1 ae 30 d6
.: 08f8 20 d2 ff d0 f6 20 d7 ca

.: 0900 38 60 a0 9d 84 ae a0 c0
.: 0908 84 af 38 e9 7f aa a0 00
.: 0910 00 ca f0 ee e6 ae d0 02
.: 0918 e6 af b1 ae 10 f6 30 f1
.: 0920 20 6b c9 a5 14 85 35 a5
.: 0928 15 85 36 20 fd ce 20 6b
.: 0930 c9 a5 14 85 33 a5 15 85
.: 0938 34 20 8e c6 20 ca ff 00

.: 0940 20 ca ff 00 d0 21 20 ac
.: 0948 ff 00 20 ca ff 00 20 ca
.: 0950 ff 00 d0 03 4c 8f fc 00
.: 0958 20 ca ff 00 a5 63 91 7a
.: 0960 20 ca ff 00 a5 62 91 7a
.: 0968 20 b7 ff 00 f0 e2 20 ca
.: 0970 ff 00 20 ca ff 00 20 ca
.: 0978 ff 00 c9 22 d0 0b 20 ca

.: 0980 ff 00 f0 c5 c9 22 d0 f7
.: 0988 f0 ee aa f0 bc 10 e9 a2
.: 0990 04 dd d4 ff 00 f0 05 ca
.: 0998 d0 f8 f0 dd a5 7a 85 3b
.: 09a0 a5 7b 85 3c 20 73 00 00
.: 09a8 b0 d3 20 6b c9 20 51 ff
.: 09b0 00 a5 3c 85 7b a5 3b 85
.: 09b8 7a a0 00 00 a2 00 00 bd

.: 09c0 00 00 01 c9 30 90 11 48
.: 09c8 20 73 00 00 90 03 20 82
.: 09d0 ff 00 68 a0 00 00 91 7a
.: 09d8 e8 d0 e8 20 73 00 00 b0
.: 09e0 08 20 91 ff 00 20 79 00
.: 09e8 00 90 f8 c9 2c f0 b8 d0
.: 09f0 96 20 ac ff 00 20 ca ff
.: 09f8 00 20 ca ff 00 d0 08 a9

.: 0a00 ff 85 63 85 62 30 0e 20
.: 0a08 ca ff 00 c5 14 d0 0f 20
.: 0a10 ca ff 00 c5 15 d0 0b 20
.: 0a18 d1 dd a9 20 4c d2 ff 20
.: 0a20 ca ff 00 20 b7 ff 00 f0
.: 0a28 d2 20 a2 ff 00 e6 97 20
.: 0a30 24 fe 00 e6 2d d0 02 e6
.: 0a38 2e 60 20 a2 ff 00 c6 97

.: 0a40 20 0c fe 00 a5 2d d0 02
.: 0a48 c6 2e c6 2d 60 20 f0 fd
.: 0a50 00 a0 00 00 84 0b 84 97
.: 0a58 60 a5 35 85 63 a5 36 85
.: 0a60 62 4c 8e c6 a5 63 18 65
.: 0a68 33 85 63 a5 62 65 34 85
.: 0a70 62 20 ca ff 00 d0 fb 60
.: 0a78 a0 00 00 e6 7a d0 02 e6

.: 0a80 7b b1 7a 60 89 8a 8d a7
.: 0a88 43 48 41 4e 47 c5 44 45
.: 0a90 4c 45 54 c5 46 49 4e c4
.: 0a98 4b 49 4c cc 4e 55 4d 42
.: 0aa0 45 d2 00 00 a5 fc 00 41
.: 0aa8 fc 00 a5 fc 00 c6 fb 00
.: 0ab0 98 fe 00 ac fb 00 aa aa
.: 0ab8 aa aa aa aa aa aa aa aa
```

ASCII Modem Driver For VIC RS–232

One very practical use of the Commodore VIC–20 is in communications. Given the sticker price of the VIC and a VICModem (or a standard RS–232 modem with the VIC RS–232 adapter), you and your budget will find this rapidly growing pastime an economical introduction to computing.

The VIC has a built–in RS232 interface. But don't go looking for a standard RS–232 connector (that trapezoid shaped connector much like The Transactor masthead border) because you won't find one. Commodore decided they could get some more mileage out of the User Port (the card–edge connector on the far right looking from the back) and make it double as RS–232 since it's unlikely both will be in use simultaneously. There's just one catch. The User Port is driven by a TTL device (known as the 6522 Versatile Interface Adapter), so output levels range from 0 to 5 volts. RS–232 modems require a –12 to +12 volt range, so the VIC RS–232 adapter (VIC 1011A) is required. The VICModem also connects to the User Port, but it has the adapter built in.

The VIC also has built–in ROM routines for transmitting and receiving characters over the RS–232. These routines do the actual handshaking of data and are invisible to the user. However, they don't do everything. Some BASIC is still necessary to get them started, interact, and then shut them off when no longer needed.

When a file is OPENed to device 2, the RS–232 port, 2 reservoirs or "buffers" are automatically allocated; one for storing characters from the keyboard and the other for characters coming in from the modem. Both buffers are 256 bytes long and are placed "back–to–back" in the topmost 512 bytes of RAM. The OPEN command also seals off this area of memory so that nothing else (like strings) will try to use it. If you plan to use RS–232, the OPEN should occur very early in your program. Otherwise, any strings that are built at the top of memory will get clobbered when the OPEN command allocates these buffers. Also make sure your program has a free space of at least 512 bytes (preferably more for variables and arrays) or you'll suddenly find that the end of your text has been gobbled!

Two other rather noteworthy events occur upon opening the RS–232. First, the serial port is disa»led. That's right, no more disk or printer. If that wasn't enough, the cassette port is also benched. It seems that the VIC devotes quite a bit of its interface internals to supporting RS–232. Disk and printer logging is still possible, however it would require closing down the RS–232 while they are serviced. These features would slow down a BASIC program to the point where characters might be lost. A machine language driver would be necessary for a more sophisticated terminal program.

The transmit/receive routines begin working when characters start appearing in either one of these buffers. Usually

only one buffer sees activity at any one time. Rarely do both ends of a communications link want to talk at the same time. So, after you send characters, you'll be waiting for a response. Don't worry about characters from both ends mixing together on any single line. It probably won't happen.

Each buffer has 2 one–byte pointers that are stored from $029B to $029E. A one–byte pointer is more accurately referred to as an "index", but we'll use the term "pointer" for better readability and clarity.

The Input or Receiver Buffer

This one collects characters coming from the modem. The first pointer ($029B) tells the t/r routines where to store the next incoming character and the other ($029C) tells BASIC where it can GET# its next character from. Initially, the 2 pointers start out at the same place. At this point a GET# would return a null string into the variable. When a character arrives at the port, it's stored in the receiver buffer and the pointer at $029B is incremented. Now there's a gap between the two pointers. The next GET# would pull this character out of the buffer and $029C is incremented, making both pointers equal again. Essentially, one pointer is chasing the another in an attempt to keep the gap as small as possible, much like a race between two vehicles on a 256 unit track.

Thus, the ideal order of events is. . . 1 character comes in, BASIC gets it out, another comes in, and so on. But such is not always the case. At very high transfer speeds (which the VIC is prepared to handle), BASIC can potentially fall behind. A character comes in, and another, and another, and BASIC may only pull one out for every 2 or 3 that come in. This situation is OK until there is 255 characters waiting for BASIC o retrieve. When the next one arrives, $029B is incremented, and the 2 pointers become equal ($092B has "lapped" $029C). Those 256 characters are now lost because it appears to the VIC that there are no characters in the buffer.

At 300 baud, approximately 30 chars per second, (a very popular speed in the micro environment) this problem is not likely to arise unless there is too much BASIC between subsequent GET#'s. If a continuous stream of data is slamming into the port, too many IF statements might slow down BASIC enough that entire buffers get dropped. Short incoming bursts should be no problem though. As long as the strings are less than 256, they'll wait in the buffer comfortably until BASIC gets them out.

The Output or Transmitter Buffer

Characters are sent from this buffer which operates much the same way as the receiver buffer. Once again, 2 pointers control the flow of data in and out of the buffer. The first, at $029D, tells the transmit/receive routines where to grab another character for sending. The other pointer, at $029E, indicates where the PRINT# command will place subsequent characters. As they arrive in the buffer, this pointer is incremented, and as they get sent, $029D is incremented. So, like the input buffer, the t/r routines are constantly trying to eliminate any gap between the two pointers.

Unlike the input buffer, the t/r routines won't allow the output buffer to overflow. If PRINT# tries to place more than 254 characters in the buffer, BASIC will wait until some of them are sent before letting any more pile up. Since most often the keyboard will be the main source of output characters, it's unlikely that the output buffer will see this condition. A send–from–disk feature could possible fill up the transmitter buffer, but, as mentioned earlier, the serial port is disabled by the RS–232, so disk is inaccessable anyways. A machine code driver would need to close the RS–232, bring in some disk material, re–open communications and continue from there. One would also have to incorporate certain precautions to take care of the buffer space that gets de–allocated and then re–allocated during this procedure. A garbage collect during disk/printer access might put some vital strings up there that would eventually be destroyed. As you can see, an all–encompassing VIC modem driver would require some careful thought.

Alias ST

The status variable ST takes on a whole new meaning during RS–232 use. Bits within ST are used to flag various transmission errors but if everything seems to be going OK, an ST test is often unecessary. If you'd like to insert one, a simple test for ST<>0 will indicate an error, otherwise smooth sailing. ST is cleared to zero immediately after being read, so if you want to know what it was, you'll need to trap it into some other variable.

Closing Up

A simple CLOSE brings your VIC back to normal. The space eaten up by those buffers is returned (try ?FRE(0)), the serial port comes back, and the cassette works once more. But before you do, Commodore suggests this line just to make sure everything's been sent:

```
140 if peek(37151) and 64 then 140
```

More VIC RS–232 Info

The VIC Programmers Reference Guide has a section on RS–232. Details on control register and command register setup is covered along with some more RS–232 facts.

COMPUTE's August '82 Issue 27 has a article worth looking up by Jim Butterfield and Jim Law. For those of you getting bored with details, here's what we've been leading up to.

ASCII Terminal Software

Below is a program that will get you started. Simpler ones are around, but this one talks true ASCII. For a more detailed description of the program, take a look at the article titled Translation Arrays in this issue.

```
100 gosub 200
110 get#1, a$ : if a$ then print     chr$(f(asc(a$)));
120 get a$     : if a$ then print#1 chr$(t(asc(a$)));
130 if a$<>"!" goto 110       : rem or subst. any char to end prog
140 if peek(37151) and 64 then 140
150 close 1
160 print chr$(9)                 : rem enables Shift/CBM
170 end
200 open 1, 2, 3, chr$(6)+ chr$(160) : rem sets 300 baud
210 print chr$(14)                : rem sets upper/lower case
220 print chr$(8) : rem locks out case change by Shift/CBM
230 dim t(255), f(255)
240 for j=32  to  64 : t(j)=j       : next
250 for j=65  to  90 : t(j)=j+32  : next
260 for j=91  to  95 : t(j)=j       : next
270 for j=192 to 218 : t(j)=j–128 : next
280 t(13)=13 : t(20)=8
290 rem add extra functions here
300 for j=0to255 :if t(j) then f(t(j))=j : f(t(j)+128)=j : next
310 return
```

The Commodore 64: A Preliminary Review

In a world where new microcomputers seem to be appearing faster than federal budgets, it's hard to decide when to buy. That fear of obsolescence is very real when YOUR money is on the line. But all computers will eventually become obsolete, so waiting for a better one is perpetual. Therefore, you must determine which computer will do everything you desire at a price that suits your budget. That's a pretty tall order, because you might find one that does everything, but a second mortgage on your house was not in your plans. Alternately, that inexpensive machine you've had your eye on might only give you room for a simple mortgage program and you find yourself faced with expanding not only the computer but your investment too. Finding a compromise can be tough.

If you've decided that you're definitely in the market for a micro, check out the Commodore 64; Commodore's newest entry and, without a doubt, their best so far. Although it looks like a VIC from the outside, inside it's a whole new story.

The 64 already has 64K of RAM (hence the name), so "memory expansion" can be scratched off your shopping list. Of course, not all 64K is available simultaneously. Memory is split into sections which must be switched in or out as required. More on 64 bank–switching in a later issue.

Like the VIC, it comes in that "wedge" shaped plastic housing, but in a colour that looks like a cross between beige and grey. Unlike the VIC, it has a 40 column by 25 line screen output and the modulator is contained inside the housing where it belongs. The keyboard feels a little nicer too, but that's only a personal opinion, possibly influenced by the order in which they came out.

Standard Design Features

- Cartridge slot for games, etc., compatible with the new MAX machine.
- 8 bit user port
- Serial bus for disk, printer, etc., (like on the VIC)
- Cassette tape port
- Composite video output and modulator output ports
- Audio output jack
- 38K available for BASIC text
- 2 built–in Analog to Digital converters
- 16 colours

And there's more. . .

Several new chips have been used in the 64's design. A new microprocessor, the MOS 6510, makes its debut. The 6510 is still an 8 bit machine, but internal bank switching capabilities allow for the bonus RAM. Other features include: pro-

grammable stack pointer, variable length stack, an 8 bit bi–directional I/O port and it's software compatible with 6502 programs.

The 6566 Video Display Chip

This one's a beauty! Everything you ever wanted to do in movable graphics. . . and then some. The 6566 has the capability of moving definable shapes, or more commonly known as "sprites" . Up until now, sprites have only been available on machines like arcade games. Pacmans, galaxians, frogs, etc, are all done using sprites.

The 64 allows 8 sprites to be displayed simultaneously (with more by using "raster compare" which we'll explain in a later issue). You simply define a shape somewhere in memory, select a sprite number, point that sprite at the shape, and turn it on. Like magic that shape will appear instantly on the screen. Then by merely giving X and Y coordinates, the shape moves around the screen by itself! No more erasing the shape at its previous location, no more updating screen RAM and colour tables. . . everything's done in hardware!

Other sprite features include: horizontal and/or vertical size expansion; hi–res or multi colour sprites; collision detection between sprites, or sprites and background; sprite/background display priority (sprites can appear in front of background or disappear behind background).

Other 6566 features include: smooth scrolling horizontally and vertically; 16 colours with 3 grey shades; bit map mode for hi–resolution display; I/O ports for 2 joysticks or 4 paddles or light pen input.

The 6581 Sound Interface Device (SID)

The SID is virtually a synthesizer on a chip. It has 3 independently controllable voices, each with a 9 octave range and 4 waveforms including square, triangle, sawtooth and noise. Each voice also has a programmable envelope generator and volume control with a master volume control for all three voices. The SID has some other very sophisticated features such as oscillator synchronization, ring modulation, filter resonance control, and it even has an external input for processing signals from other sources such as an electric guitar.

In all, the 6581 can produce sound with better quality than some of the instruments it can simulate. With a little programming, voice synthesis shouldn't be too much trouble either.

The 6526 Complex Interface Adapter (CIA)

The 64 comes with 2 of these. The 6526's replace the 6520 and 6522 of earlier machines. Each chip has an 8 bit shift register for serial I/O, 24 clock with programmable alarm, 8 or 16 bit handshaking on read or write, 2 independent 16 bit interval timers and the capability for sourcing or sinking 2 standard TTL loads. I've been told that the external input to the SID can only process signals through the filter section which eliminates several possibilities. However, the 6526 has two analog to digital converters. By connecting external sources here, audio signals can be sent through all sections of the SID which will provide for some rather interesting experimentation. The A/D's will also eliminate a lot of the analog interfacing problems that have plagued us in the past.

Accessories

Commodore intends to use many of the same accessories for the 64 as are available now for the VIC. VIC joysticks, paddles, disk drive, printer, modem, RS–232 cartridge and C2N cassette are all compatible with the 64. About the only difference is the slot for things like games and utility cartridges. It's been changed to a vertical pin type connector as opposed to the card edge type connector on the VIC–20. This is not only less space consuming, but promises to be a little more rugged as the contacts appear to be less susceptible to friction wear.

Future accessories, according to Commodore, include a Z–80 plug–in card, soft–load modules for BASIC 4.0, Pascal, Forth, and Pilot, extended BASIC cartridges for graphics and sound support, and monitor type cartridges for machine language exercises.

Some Comments

This time Commodore's done it right! They placed the screen RAM immediately following zero page, the stack, and RAM allocated for the cassette buffer. This leaves a 38K stretch of uninterrupted memory which has been set up for BASIC text. Unlike the VIC, the screen won't be moving around on you, but like the VIC, the LOAD command will adjust for the new start of text address, $0800.

The character generator is set up like in PET/CBMs. It takes no address space away from the processor unless you want to modify it. Using a very simple program, it can be transferred to RAM (where it *will* require address space) and a character pointer invokes the new location.

The 64 has 16 colours, 8 more than the VIC. Commodore

had some trouble with colour quality on TV and monitor output, but this has been cleared up in the release versions. It has three distinct grey shades that come up beautifully even through modulated output to a TV. They also have an element of brilliance about them that make them appear more like colours as opposed to just shades.

The Control key is always a nice feature on any machine. The VIC Control key apparently had some problems when used in a communications environment, but this is all cleared up for the 64. RUN/STOP–RESTORE will still get you out of just about any crash, and changing from Upper/Lower case to Graphics mode is a snap with the shifted Commodore key at the lower left corner of the keyboard. 8 Function keys are included (4 plus 4 shifted) and I've been informed that a new method of keyboard scanning will allow the Control and Commodore keys to also be used as shift keys. Combinations of CTRL, Shift, and the Commodore key might result in as many as 32 effective function keys!

Commodore opted to stay with the serial bus for interfacing peripherals. Unlike IEEE parallel, bits are delivered in serial which reduces communications speed considerably. This reduction isn't really noticeable with the printer, but disk access is somewhat hampered. Fortunately there will be an IEEE interface card available for those that already own IEEE peripherals (see the New Products section for information on RTC's V–Link).

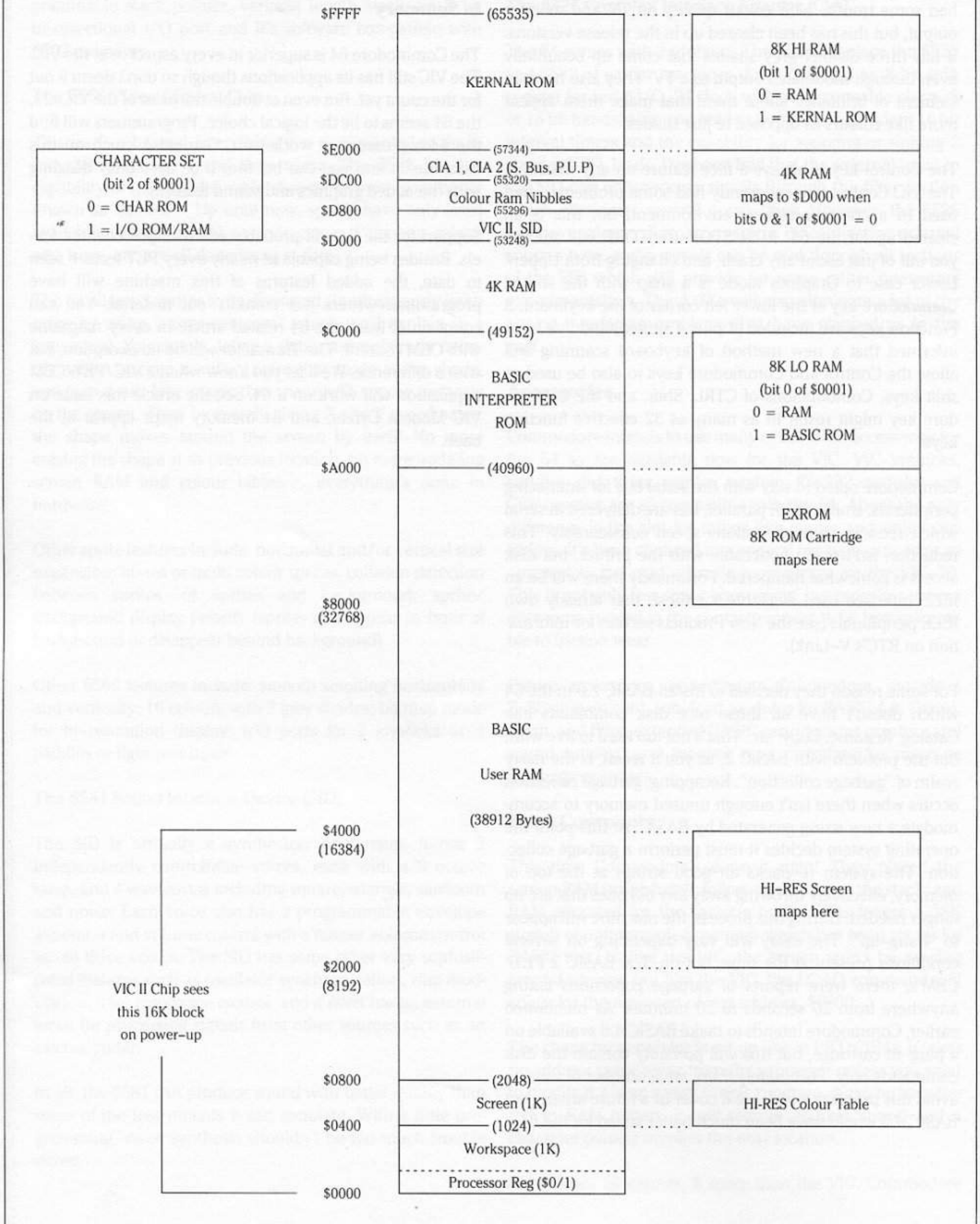
For some reason they decided to install BASIC 2.0 in the 64 which doesn't have all those nice disk commands like Catalog, Rename, Copy, etc. That's not too hard to live with, but the problem with BASIC 2, as you'll recall, is the nasty realm of "garbage collection". Recapping, garbage collection occurs when there isn't enough unused memory to accommodate a new string generated by BASIC. At this point the operating system decides it must perform a garbage collection. The system re–packs all good strings at the top of memory, effectively throwing away any old ones that are no longer needed. During this process, the machine will appear to "hang–up". The delay will vary depending on several conditions present at the time, but on 32K BASIC 2 PET/CBM's, there were reports of garbage collections taking anywhere from 20 seconds to 20 minutes. As mentioned earlier, Commodore intends to make BASIC 4.0 available on a plug–in cartridge, but this will probably contain the disk commands only. Fortunately there are a number of ways to avoid this problem which we'll cover in a future article, but BASIC 4.0 would have been much better suited for the 64.

In Summary

The Commodore 64 is superior in every aspect over the VIC. The VIC still has its applications though so don't deem it out for the count yet. But even at double the price of the VIC–20, the 64 seems to be the logical choice. Programmers will find the 64 a pleasure to work with. Games? A cinch on this machine. . . and you can bet they'll be absolutely dazzling with the added graphics and sound features.

Support for the 64 will probably reach unprecedented levels. Besides being capable of nearly every PET feature seen to date, the added features of this machine will have programmer/writers just crankin' out material. You can count on at least one 64 related article in every magazine with CBM content. The Transactor will be no exception, but with a difference. We'll let you know when a VIC/PET/CBM application will work on a 64. See the article this issue on VIC Modem Driver, and 64 memory maps appear at the back.

Commodore–64 Architecture Map



Commodore 64 Memory Map

Compiled by
Jim Butterfield

0000		0	Chip directional register
0001		1	Chip I/O; memory & tape control
0003	-0004	3-4	Float-Fixed vector
0005	-0006	5-6	Fixed-Float vector
0007		7	Search character
0008		8	Scan-quotes flag
0009		9	TAB column save
000A		10	0 = LOAD, 1 = VERIFY
000B		11	Input buffer pointer/# subscrpt
000C		12	Default DIM flag
000D		13	Type: FF = string, 00 = numeric
000E		14	Type: 80 = integer, 00 = floating point
000F		15	DATA scan/LIST quote/memry flag
0010		16	Subscript/FNx flag
0011		17	0 = INPUT;$40 = GET;$98 = READ
0012		18	ATN sign/Comparison eval flag
0013		19	Current I/O prompt flag
0014	-0015	20-21	Integer value
0016		22	Pointer: temporary string stack
0017	-0018	23-24	Last temp string vector
0019	-0021	25-33	Stack for temporary strings
0022	-0025	34-37	Utility pointer area
0026	-002A	38-42	Product area for multiplication
002B	-002C	43-44	Pointer: Start-of-Basic
002D	-002E	45-46	Pointer: Start-of-Variables
002F	-0030	47-48	Pointer: Start-of-Arrays
0031	-0032	49-50	Pointer: End-of-Arrays
0033	-0034	51-52	Pointer: String-storage(moving down)
0035	-0036	53-54	Utility string pointer
0037	-0038	55-56	Pointer: Limit-of-memory
0039	-003A	57-58	Current Basic line number
003B	-003C	59-60	Previous Basic line number
003D	-003E	61-62	Pointer: Basic statement for CONT
003F	-0040	63-64	Current DATA line number
0041	-0042	65-66	Current DATA address
0043	-0044	67-68	Input vector
0045	-0046	69-70	Current variable name
0047	-0048	71-72	Current variable address
0049	-004A	73-74	Variable pointer for FOR/NEXT
004B	-004C	75-76	Y-save; op-save; Basic pointer save
004D		77	Comparison symbol accumulator
004E	-0053	78-83	Misc work area, pointers, etc
0054	-0056	84-86	Jump vector for functions
0057	-0060	87-96	Misc numeric work area
0061		97	Accum#1: Exponent
0062	-0065	98-101	Accum#1: Mantissa
0066		102	Accum#1: Sign
0067		103	Series evaluation constant pointer
0068		104	Accum#1 hi-order (overflow)
0069	-006E	105-110	Accum#2: Exponent, etc.
006F		111	Sign comparison, Acc#1 vs #2

0070		112	Accum#1 lo-order (rounding)
0071	-0072	113-114	Cassette buff len/Series pointer
0073	-008A	115-138	CHRGET subroutine; get Basic char
007A	-007B	122-123	Basic pointer (within subrtn)
008B	-008F	139-143	RND seed value
0090		144	Status word ST
0091		145	Keyswitch PIA: STOP and RVS flags
0092		146	Timing constant for tape
0093		147	Load=0, Verify=1
0094		148	Serial output: deferred char flag
0095		149	Serial deferred character
0096		150	Tape EOT received
0097		151	Register save
0098		152	How many open files
0099		153	Input device, normally 0
009A		154	Output CMD device, normally 3
009B		155	Tape character parity
009C		156	Byte-received flag
009D		157	Direct=$80/RUN=0 output control
009E		158	Tp Pass 1 error log/char buffer
009F		159	Tp Pass 2 err log corrected
00A0	-00A2	160-162	Jiffy Clock HML
00A3		163	Serial bit count/EOI flag
00A4		164	Cycle count
00A5		165	Countdown,tape write/bit count
00A6		166	Tape buffer pointer
00A7		167	Tp Wrt ldr count/Rd pass/inbit
00A8		168	Tp Wrt new byte/Rd error/inbit cnt
00A9		169	Wrt start bit/Rd bit err/stbit
00AA		170	Tp Scan;Cnt;Ld;End/byte assy
00AB		171	Wr lead length/Rd checksum/parity
00AC	-00AD	172-173	Pointer: tape bufr, scrolling
00AE	-00AF	174-175	Tape end adds/End of program
00B0	-00B1	176-177	Tape timing constants
00B2	-00B3	178-179	Pntr: start of tape buffer
00B4		180	1=Tp timer enabled; bit count
00B5		181	Tp EOT/RS232 next bit to send
00B6		182	Read character error/outbyte buf
00B7		183	# characters in file name
00B8		184	Current logical file
00B9		185	Current secndy address
00BA		186	Current device
00BB	-00BC	187-188	Pointer to file name
00BD		189	Wr shift word/Rd input char
00BE		190	# blocks remaining to Wr/Rd
00BF		191	Serial word buffer
00C0		192	Tape motor interlock
00C1	-00C2	193-194	I/O start address
00C3	-00C4	195-196	Kernel setup pointer
00C5		197	Last key pressed
00C6		198	# chars in keybd buffer
00C7		199	Screen reverse flag
00C8		200	End-of-line for input pointer
00C9	-00CA	201-202	Input cursor log (row, column)
00CB		203	Which key: 64 if no key

00CC		204	0 = flash cursor
00CD		205	Cursor timing countdown
00CE		206	Character under cursor
00CF		207	Cursor in blink phase
00D0		208	Input from screen/from keyboard
00D1	-00D2	209-210	Pointer to screen line
00D3		211	Position of cursor on above line
00D4		212	0 = direct cursor, else programmed
00D5		213	Current screen line length
00D6		214	Row where curosr lives
00D7		215	Last inkey/checksum/buffer
00D8		216	# of INSERTs outstanding
00D9	-00F2	217-242	Screen line link table
00F3	-00F4	243-244	Screen color pointer
00F5	-00F6	245-246	Keyboard pointer
00F7	-00F8	247-248	RS-232 Rcv pntr
00F9	-00FA	249-250	RS-232 Tx pntr
00FF	-010A	256-266	Floating to ASCII work area
0100	-013E	256-318	Tape error log
0100	-01FF	256-511	Processor stack area
0200	-0258	512-600	Basic input buffer
0259	-0262	601-610	Logical file table
0263	-026C	611-620	Device # table
026D	-0276	621-630	Sec Adds table
0277	-0280	631-640	Keybd buffer
0281	-0282	641-642	Start of Basic Memory
0283	-0284	643-644	Top of Basic Memory
0285		645	Serial bus timeout flag
0286		646	Current color code
0287		647	Color under cursor
0288		648	Screen memory page
0289		649	Max size of keybd buffer
028A		650	Repeat all keys
028B		651	Repeat speed counter
028C		652	Repeat delay counter
028D		653	Keyboard Shift/Control flag
028E		654	Last shift pattern
028F	-0290	655-656	Keyboard table setup pointer
0291		657	Keyboard shift mode
0292		658	0 = scroll enable
0293		659	RS-232 control reg
0294		660	RS-232 command reg
0295	-0296	661-662	Bit timing
0297		663	RS-232 status
0298		664	# bits to send
0299	-029A	665	RS-232 speed/code
029B		667	RS232 receive pointer
029C		668	RS232 input pointer
029D		669	RS232 transmit pointer
029E		670	RS232 output pointer
029F	-02A0	671-672	IRQ save during tape I/O
02A1		673	CIA 2 (NMI) Interrupt Control
02A2		674	CIA 1 Timer A control log
02A3		675	CIA 1 Interrupt Log
02A4		676	CIA 1 Timer A enabled flag

02A5		677	Screen row marker	
02C0	-02FE	704-766	(Sprite 11)	
0300	-0301	768-769	Error message link	
0302	-0303	770-771	Basic warm start link	
0304	-0305	772-773	Crunch Basic tokens link	
0306	-0307	774-775	Print tokens link	
0308	-0309	776-777	Start new Basic code link	
030A	-030B	778-779	Get arithmetic element link	
030C		780	SYS A-reg save	
030D		781	SYS X-reg save	
030E		782	SYS Y-reg save	
030F		783	SYS status reg save	
0310	-0312	784-785	USR function jump	(B248)
0314	-0315	788-789	Hardware interrupt vector	(EA31)
0316	-0317	790-791	Break interrupt vector	(FE66)
0318	-0319	792-793	NMI interrupt vector	(FE47)
031A	-031B	794-795	OPEN vector	(F34A)
031C	-031D	796-797	CLOSE vector	(F291)
031E	-031F	798-799	Set-input vector	(F20E)
0320	-0321	800-801	Set-output vector	(F250)
0322	-0323	802-803	Restore I/O vector	(F333)
0324	-0325	804-805	INPUT vector	(F157)
0326	-0327	806-807	Output vector	(F1CA)
0328	-0329	808-809	Test-STOP vector	(F6ED)
032A	-032B	810-811	GET vector	(F13E)
032C	-032D	812-813	Abort I/O vector	(F32F)
032E	-032F	814-815	Warm start vector	(FE66)
0330	-0331	816-817	LOAD link	(F4A5)
0332	-0333	818-819	SAVE link	(F5ED)
033C	-03FB	828-1019	Cassette buffer	
0340	-037E	832-894	(Sprite 13)	
0380	-03BE	896-958	(Sprite 14)	
03C0	-03FE	960-1022	(Sprite 15)	
0400	-07FF	1024-2047	Screen memory	
0800	-9FFF	2048-40959	Basic RAM memory	
8000	-9FFF	32768-40959	Alternate: ROM plug-in area	
A000	-BFFF	40960-49151	ROM: Basic	
A000	-BFFF	49060-59151	Alternate: RAM	
C000	-CFFF	49152-53247	RAM memory, including alternate	
D000	-D02E	53248-53294	Video Chip (6566)	
D400	-D41C	54272-54300	Sound Chip (6581 SID)	
D800	-DBFF	55296-56319	Color nybble memory	
DC00	-DC0F	56320-56335	Interface chip 1, IRQ (6526 CIA)	
DD00	-DD0F	56576-56591	Interface chip 2, NMI (6526 CIA)	
D000	-DFFF	53248-53294	Alternate: Character set	
E000	-FFFF	57344-65535	ROM: Operating System	
E000	-FFFF	57344-65535	Alternate: RAM	
FF81	-FFF5	65409-65525	Jump Table, Including:	
FFC6			- Set Input channel	
FFC9			- Set Output channel	
FFCC			- Restore default I/O channels	
FFCF			- INPUT	
FFD2			- PRINT	
FFE1			- Test Stop key	
FFE4			- GET	

Commodore 64 – ROM Memory Map

A000; ROM control vectors
A00C; Keyword action vectors
A052; Function vectors
A080; Operator vectors
A09E; Keywords
A19E; Error messages
A328; Error message vectors
A365; Misc messages
A38A; Scan stack for FOR/GOSUB
A3B8; Move memory
A3FB; Check stack depth
A408; Check memory space
A435; 'out of memory'
A437; Error routine
A469; BREAK entry
A474; 'ready.'
A480; Ready for Basic
A49C; Handle new line
A533; Re–chain lines
A560; Receive input line
A579; Crunch tokens
A613; Find Basic line
A642; Perform [NEW]
A65E; Perform [CLR]
A68E; Back up text pointer
A69C; Perform [LIST]
A742; Perform [FOR]
A7ED; Execute statement
A81D; Perform [RESTORE]
A82C; Break
A82F; Perform [STOP]
A831; Perform [END]
A857; Perform [CONT]
A871; Perform [RUN]
A883; Perform [GOSUB]
A8A0; Perform [GOTO]
A8D2; Perform [RETURN]
A8F8; Perform [DATA]
A906; Scan for next statement
A928; Perform [IF]
A93B; Perform [REM]
A94B; Perform [ON]
A96B; Get fixed point number
A9A5; Perform [LET]
AA80; Perform [PRINT#]
AA86; Perform [CMD]
AAA0; Perform [PRINT]
AB1E; Print string from (y.a)
AB3B; Print format character
AB4D; Bad input routine
AB7B; Perform [GET]
ABA5; Perform [INPUT#]
ABBF; Perform [INPUT]
ABF9; Prompt & input
AC06; Perform [READ]
ACFC; Input error messages
AD1E; Perform [NEXT]
AD78; Type match check
AD9E; Evaluate expression
AEA8; Constant – pi
AEF1; Evaluate within brackets
AEF7; ')'
AEFF; comma..
AF08; Syntax error
AF14; Check range
AF28; Search for variable
AFA7; Setup FN reference
AFE6; Perform [OR]
AFE9; Perform [AND]
B016; Compare
B081; Perform [DIM]
B08B; Locate variable
B113; Check alphabetic
B11D; Create variable
B194; Array pointer subrtine
B1A5; Value 32768
B1B2; Float–fixed
B1D1; Set up array
B245; 'bad subscript'
B248; 'illegal quantity'
B34C; Compute array size
B37D; Perform [FRE]
B391; Fix–float
B39E; Perform [POS]
B3A6; Check direct
B3B3; Perform [DEF]
B3E1; Check fn syntax
B3F4; Perform [FN]
B465; Perform [STR$]
B475; Calculate string vector
B487; Set up string
B4F4; Make room for string
B526; Garbage collection
B5BD; Check salvageability
B606; Collect string
B63D; Concatenate
B67A; Build string to memory
B6A3; Discard unwanted string
B6DB; Clean descriptor stack
B6EC; Perform [CHR$]
B700; Perform [LEFT$]
B72C; Perform [RIGHT$]
B737; Perform [MID$]
B761; Pull string parameters
B77C; Perform [LEN]
B782; Exit string–mode
B78B; Perform [ASC]
B79B; Input byte paramter
B7AD; Perform [VAL]
B7EB; Parameters for POKE/WAIT
B7F7; Float–fixed
B80D; Perform [PEEK]
B824; Perform [POKE]
B82D; Perform [WAIT]

B849; Add 0.5
B850; Subtract-from
B853; Perform [subtract]
B86A; Perform [add]
B947; Complement FAC#1
B97E; 'overflow'
B983; Multiply by zero byte
B9EA; Perform [LOG]
BA2B; Perform [multiply]
BA59; Multiply-a-bit
BA8C; Memory to FAC#2
BAB7; Adjust FAC#1/#2
BAD4; Underflow/overflow
BAE2; Multiply by 10
BAF9; +10 in floating pt
BAFE; Divide by 10
BB12; Perform [divide]
BBA2; Memory to FAC#1
BBC7; FAC#1 to memory
BBFC; FAC#2 to FAC#1
BC0C; FAC#1 to FAC#2
BC1B; Round FAC#1
BC2B; Get sign
BC39; Perform [SGN]
BC58; Perform [ABS]
BC5B; Compare FAC#1 to mem
BC9B; Float-fixed
BCCC; Perform [int]
BCF3; String to FAC
BD7E; Get ascii digit
BDC2; Print 'IN..'
BDCD; Print line number
BDDD; Float to ascii
BF16; Decimal constants
BF3A; TI constants
BF71; Perform [SQR]
BF7B; Perform [power]
BFB4; Perform [negative]
BFED; Perform [EXP]
E043; Series eval 1
E059; Series eval 2
E097; Perform [RND]
E0f9; ?? breakpoints ??
E12A; Perform [SYS]
E156; Perform [SAVE]
E165; Perform [VERIFY]
E168; Perform [LOAD]
E1BE; Perform [OPEN]
E1C7; Perform [CLOSE]
E1D4; Parameters for LOAD/SAVE
E206; Check default parameters
E20E; Check for comma
E219; Parameters for open/close
E264; Perform [COS]
E26B; Perform [SIN]
E2B4; Perform [TAN]
E30E; Perform [ATN]
E37B; Warm restart

E394; Initialize
E3A2; CHRGET for zero page
E3BF; Initialize Basic
E447; Vectors for $300
E453; Initialize vectors
E45F; Power-up message
E500; Get I/O address
E505; Get screen size
E50A; Put/get row/column
E518; InitializeI/O
E544; Clear screen
E566; Home cursor
E56C; Set screen pointers
E5A0; Set I/O defaults
E5B4; Input from keyboard
E632; Input from screen
E684; Quote test
E691; Setup screen print
E6B6; Advance cursor
E6ED; Retreat cursor
E701; Back into previous line
E716; Output to screen
E87C; Go to next line
E891; Perform <return>
E8A1; Check line decrement
E8B3; Check line increment
E8CB; Set color code
E8DA; Color code table
E8EA; Scroll screen
E965; Open space on screen
E9C8; Move a screen line
E9E0; Synchronize color transfer
E9F0; Set start-of-line
E9FF; Clear screen line
EA13; Print to screen
EA24; Synchronize color pointer
EA31; Interrupt - clock etc
EA87; Read keyboard
EB79; Keyboard select vectors
EB81; Keyboard 1 - unshifted
EBC2; Keyboard 2 - shifted
EC03; Keyboard 3 - 'comm'
EC44; Graphics/text contrl
EC4F; Set graphics/text mode
EC78; Keyboard 4
ECB9; Video chip setup
ECE7; Shift/run equivalent
ECF0; Screen ln address low
ED09; Send 'talk'
ED0C; Send 'listen'
ED40; Send to serial bus
EDB2; Serial timeout
EDB9; Send listen SA
EDBE; Clear ATN
EDC7; Send talk SA
EDCC; Wait for clock
EDDD; Send serial deferred
EDEF; Send 'untalk'

EDFE; Send 'unlisten'
EE13; Receive from serial bus
EE85; Serial clock on
EE8E; Serial clock off
EE97; Serial output '1'
EEA0; Serial output '0'
EEA9; Get serial in & clock
EEB3; Delay 1 ms
EEBB; RS–232 send
EF06; Send new RS–232 byte
EF2E; No–DSR error
EF31; No–CTS error
EF3B; Disable timer
EF4A; Compute bit count
EF59; RS232 receive
EF7E; Setup to receive
EFC5; Receive parity error
EFCA; Receive overflow
EFCD; Receive break
EFD0; Framing error
EFE1; Submit to RS232
F00D; No–DSR error
F017; Send to RS232 buffer
F04D; Input from RS232
F086; Get from RS232
F0A4; Check serial bus idle
F0BD; Messages
F12B; Print if direct
F13E; Get..
F14E; ..from RS232
F157; Input
F199; Get.. tape/serial/rs232
F1CA; Output..
F1DD; ..to tape
F20E; Set input device
F250; Set output device
F291; Close file
F30F; Find file
F31F; Set file values
F32F; Abort all files
F333; Restore default I/O
F34A; Do file open
F3D5; Send SA
F409; Open RS232
F49E; Load program
F5AF; 'searching'
F5C1; Print filename
F5D2; 'loading/verifying'
F5DD; Save program
F68F; Print 'saving'
F69B; Bump clock
F6BC; Log PIA key reading
F6DD; Get time
F6E4; Set time
F6ED; Check stop key
F6FB; Output error messages
F72D; Find any tape headr
F76A; Write tape header
F7D0; Get buffer address
F7D7; Set buffer start/end pointers
F7EA; Find specific header
F80D; Bump tape pointer
F817; 'press play..'
F82E; Check tape status
F838; 'press record..'
F841; Initiate tape read
F864; Initiate tape write
F875; Common tape code
F8D0; Check tape stop
F8E2; Set read timing
F92C; Read tape bits
FA60; Store tape chars
FB8E; Reset pointer
FB97; New character setup
FBA6; Send transition to tape
FBC8; Write data to tape
FBCD; IRQ entry point
FC57; Write tape leader
FC93; Restore normal IRQ
FCB8; Set IRQ vector
FCCA; Kill tape motor
FCD1; Check r/w pointer
FCDB; Bump r/w pointer
FCE2; Power reset entry
FD02; Check 8–rom
FD10; 8–rom mask
FD15; Kernal reset
FD1A; Kernal move
FD30; Vectors
FD50; Initialize system constnts
FD9B; IRQ vectors
FDA3; Initialize I/O
FDDD; Enable timer
FDF9; Save filename data
FE00; Save file details
FE07; Get status
FE18; Flag status
FE1C; Set status
FE21; Set timeout
FE25; Read/set top of memory
FE27; Read top of memory
FE2D; Set top of memory
FE34; Read/set bottom of memory
FE43; NMI entry
FE66; Warm start
FEB6; Reset IRQ & exit
FEBC; Interrupt exit
FEC2; RS–232 timing table
FED6; NMI RS–232 in
FF07; NMI RS–232 out
FF43; Fake IRQ
FF48; IRQ entry
FF81; Jumbo jump table
FFFA; Hardware vectors

Processor I/O Port (6510)

$0000	IN	IN	OUT	IN	OUT	OUT	OUT	OUT	DDR 0
$0001			Tape Motor	Tape Sense	Tape Write	D-ROM Switch	EF RAM Switch	AB RAM Switch	PR 1

SID (6581)

Voice 1	Voice 2	Voice 3		Voice 1	Voice 2	Voice 3
$D400	$D407	$D40E	Frequency L	54272	54279	54286
$D401	$D408	$D40F	Frequency H	54273	54280	54287
$D402	$D409	$D410	Pulse Width L	54274	54281	54288
$D403	$D40A	$D411	0 0 0 0 H	54275	54282	54289
$D404	$D40B	$D412	Voice Type: NSE PUL SAW TRI Key	54276	54283	54290
$D405	$D40C	$D413	Attack Time 2ms – 8ms / Decay Time 6ms – 24 sec	54277	54284	54291
$D406	$D40D	$D414	Sustain Level / Release Time 6ms 24 sec	54278	54285	54292

Voices (write only)

$D415	0 0 0 0 0 L	54293
$D416	Filter Frequency H	54294
$D417	Resonance / Filter Voices: Ext V3 V2 V1	54295
$D418	V3 off / Passband: HI BP LO / Master Volume	54296

Filter & Volume (write only)

$D419	Paddle X (A/D #1)	54297
$D41A	Paddle Y (A/D #2)	54298
$D41B	Noise 3 (random)	54299
$D41C	Envelope 3	54300

Sense (read only)

Note: Special Voice Features
(TEST, RING MOD, SYNC)
are omitted from the above diagram.

CIA 1 (IRQ) (6526)

$DC00	Paddle Sel A	Paddle Sel B		Joystick 0 Fire	Joystick 0 Right	Joystick 0 Left	Joystick 0 Down	Joystick 0 Up	PRA 56320
	Keyboard Row Select (inverted)								
$DC01				Joystick 1 Fire	Joystick 1 Right	Joystick 1 Left	Joystick 1 Down	Joystick 1 Up	PRB 56321
	Keyboard Column Read								
$DC02	$FF – All Output								DDRA 56322
$DC03	$00 – All Input								DDRB 56323
$DC04	Timer A								TAL 56324
$DC05									TAH 56325
$DC06	Timer B								TBL 56326
$DC07									TBH 56327
$DC0D			Tape Input			Timer Interrupt B	Timer Interrupt A		ICR 56333
$DC0E				One Shot	Out Mode	Time PB6 Out	Timer A Start		CRA 56334
$DC0F				One Shot	Out Mode	Time PB7 Out	Timer B Start		CRB 56335

CIA 2 (NMI) (6526)

$DD00	Serial IN	Clock IN	Serial OUT	Clock OUT	ATN OUT	RS-232 OUT	VIC II addr 15	VIC II addr 14	PRA 56576
$DD01	DSR IN	CTS IN		DCD* IN	RI* IN	DTR OUT	RTS OUT	RS-232 IN	PRB 56577
$DD02	$3F – Serial								DDRA 56578
$DD03	$00 – P.U.P. All Input			or		$06 – RS-232			DDRB 56579
$DD04	Timer A								TAL 56580
$DD05									TAH 56581
$DD06	Timer B								TBL 56582
$DD07									TBH 56583
$DD0D			RS-232 IN			Timer Interrupt B	Timer Interrupt A		ICR 56589
$DD0E							Timer A Start		CRA 56590
$DD0F							Timer B Start		CRB 56591

* Connected but not used by O.S.

Commodore 64 Memory Map

Compiled by Jim Butterfield

Commodore 64 – RAM Memory Map

Hex	Decimal	Description
0000	0	Chip directional register
0001	1	Chip I/O, memory & tape control
0003–0004	3–4	Float–Fixed vector
0005–0006	5–6	Fixed–Float vector
0007	7	Search character
0008	8	Scan–quotes flag
0009	9	TAB column save
000A	10	0 = LOAD, 1 = VERIFY
000B	11	Input buffer pointer/* subscript
000C	12	Default DIM flag
000D	13	Type: FF = string, 00 = numeric
000E	14	Type: 80 = integer, 00 = floating point
000F	15	DATA scan/LIST quote/memory flag
0010	16	Subscript/FNx flag
0011	17	0 = INPUT;$40 = GET;$98 = READ
0012	18	ATN sign/Comparison eval flag
0013	19	Current I/O prompt flag
0014–0015	20–21	Integer value
0016	22	Pointer: temporary string stack
0017–0018	23–24	Last temp string vector
0019–0021	25–33	Stack for temporary strings
0022–0025	34–37	Utility pointer area
0026–002A	38–42	Product area for multiplication
002B–002C	43–44	Pointer: Start–of–Basic
002D–002E	45–46	Pointer: Start–of–Variables
002F–0030	47–48	Pointer: Start–of–Arrays
0031–0032	49–50	Pointer: End–of–Arrays
0033–0034	51–52	Pointer: String–storage(moving down)
0035–0036	53–54	Utility string pointer
0037–0038	55–56	Pointer: Limit–of–memory
0038–003A	57–58	Current Basic line number
003B–003C	59–60	Previous Basic line number
003D–003E	61–62	Pointer: Basic statement for CONT
003F–0040	63–64	Current DATA line number
0041–0042	65–66	Current DATA address
0043–0044	67–68	Input vector
0045–0046	69–70	Current variable name
0047–0048	71–72	Current variable address
0049–004A	73–74	Variable pointer for FOR/NEXT
004B–004C	75–76	Y–save; op–save; Basic pointer save
004D	77	Comparison symbol accumulator
004E–0053	78–83	Misc work area, pointers, etc
0054–0056	84–86	Jump vector for functions
0057–0060	87–96	Misc numeric work area
0061	97	Accum*1: Exponent
0062–0065	98–101	Accum*1: Mantissa
0066	102	Accum*1: Sign
0067	103	Series evaluation constant pointer
0068	104	Accum*1 hi–order (overflow)
0069–006E	105–110	Accum*2: Exponent, etc.
006F	111	Sign comparison, Acc*1 vs *2
0070	112	Accum*1 lo–order (rounding)
0071–0072	113–114	Cassette buff len/Series pointer
0073–008A	115–138	CHRGET subroutine; get Basic char
007A–007B	122–123	Basic pointer (within subrtn)
008B–008F	139–143	RND seed value
0090	144	Status word ST
0091	145	Keyswitch PIA: STOP and RVS flags
0092	146	Timing constant for tape
0093	147	Load = 0, Verify = 1
0094	148	Serial output: deferred char flag
0095	149	Serial deferred character
0096	150	Tape EOT received
0097	151	Register save
0098	152	How many open files
0099	153	Input device, normally 0
009A	154	Output CMD device, normally 3
009B	155	Tape character parity
009C	156	Byte–received flag
009D	157	Direct = $80/RUN = 0 output control
009E	158	Tp Pass 1 error log/char buffer
009F	159	Tp Pass 2 err log corrected
00A0–00A2	160–162	Jiffy Clock HML
00A3	163	Serial bit count/EOI flag
00A4	164	Cycle count
00A5	165	Countdown,tape write/bit count
00A6	166	Tape buffer pointer
00A7	167	Tp Wrt ldr count/Rd pass/inbit
00A8	168	Tp Wrt new byte/Rd error/inbit cnt
00A9	169	Wrt start bit/Rd bit err/stbit
00AA	170	Tp Scan;Cnt;Ld;End/byte assy
00AB	171	Wr lead length/Rd checksum/parity
00AC–00AD	172–173	Pointer: tape bufr, scrolling
00AE–00AF	174–175	Tape end adds/End of program
00B0–00B1	176–177	Tape timing constants
00B2–00B3	178–179	Pntr: start of tape buffer
00B4	180	1 = Tp timer enabled; bit count
00B5	181	Tp EOT/RS232 next bit to send
00B6	182	Read character error/outbyte buf
00B7	183	* characters in file name
00B8	184	Current logical file
00B9	185	Current secndy address
00BA	186	Current device
00BB–00BC	187–188	Pointer to file name
00BD	189	Wr shift word/Rd input char
00BE	190	* blocks remaining to Wr/Rd
00BF	191	Serial word buffer
00C0	192	Tape motor interlock
00C1–00C2	193–194	I/O start address
00C3–00C4	195–196	Kernel setup pointer
00C5	197	Last key pressed
00C6	198	* chars in keybd buffer
00C7	199	Screen reverse flag
00C8	200	End–of–line for input pointer
00C9–00CA	201–202	Input cursor log (row, column)
00CB	203	Which key: 64 if no key
00CC	204	0 = flash cursor
00CD	205	Cursor timing countdown
00CE	206	Character under cursor
00CF	207	Cursor in blink phase
00D0	208	Input from screen/from keyboard
00D1–00D2	209–210	Pointer to screen line
00D3	211	Position of cursor on above line
00D4	212	0 = direct cursor, else programmed
00D5	213	Current screen line length
00D6	214	Row where cursor lives
00D7	215	Last inkey/checksum/buffer
00D8	216	* of INSERTs outstanding
00D9–00F2	217–242	Screen line link table
00F3–00F4	243–244	Screen color pointer
00F5–00F6	245–246	Keyboard pointer
00F7–00F8	247–248	RS–232 Rcv pntr
00F9–00FA	249–250	RS–232 Tx pntr
00FF–010A	256–266	Floating to ASCII work area
0100–013E	256–318	Tape error log
0100–01FF	256–511	Processor stack area
0200–0258	512–600	Basic input buffer
0259–0262	601–610	Logical file table
0263–026C	611–620	Device * table
026D–0276	621–630	Sec Adds table
0277–0280	631–640	Keybd buffer
0281–0282	641–642	Start of Basic Memory
0283–0284	643–644	Top of Basic Memory
0285	645	Serial bus timeout flag
0286	646	Current color code
0287	647	Color under cursor
0288	648	Screen memory page
0289	649	Max size of keybd buffer
028A	650	Repeat all keys
028B	651	Repeat speed counter
028C	652	Repeat delay counter
028D	653	Keyboard Shift/Control flag
028E	654	Last shift pattern
028F–0290	655–656	Keyboard table setup pointer
0291	657	Keyboard shift mode
0292	658	0 = scroll enable
0293	659	RS–232 control reg
0294	660	RS–232 command reg
0295–0296	661–662	Bit timing
0297	663	RS–232 status
0298	664	* bits to send
0299–029A	665	RS–232 speed/code
029B	667	RS232 receive pointer
029C	668	RS232 input pointer
029D	669	RS232 transmit pointer
029E	670	RS232 output pointer
029F–02A0	671–672	IRQ save during tape I/O
02A1	673	CIA 2 (NMI) Interrupt Control
02A2	674	CIA 1 Timer A control log
02A3	675	CIA 1 Interrupt Log
02A4	676	CIA 1 Timer A enabled flag
02A5	677	Screen row marker
02C0–02FE	704–766	(Sprite 11)
0300–0301	768–769	Error message link
0302–0303	770–771	Basic warm start link
0304–0305	772–773	Crunch Basic tokens link
0306–0307	774–775	Print tokens link
0308–0309	776–777	Start new Basic code link
030A–030B	778–779	Get arithmetic element link
030C	780	SYS A–reg save
030D	781	SYS X–reg save
030E	782	SYS Y–reg save
030F	783	SYS status reg save
0310–0312	784–785	USR function jump (B248)
0314–0315	788–789	Hardware interrupt vector (EA31)
0316–0317	790–791	Break interrupt vector (FE66)
0318–0319	792–793	NMI interrupt vector (FE47)
031A–031B	794–795	OPEN vector (F34A)
031C–031D	796–797	CLOSE vector (F291)
031E–031F	798–799	Set–input vector (F20E)
0320–0321	800–801	Set–output vector (F250)
0322–0323	802–803	Restore I/O vector (F333)
0324–0325	804–805	INPUT vector (F157)
0326–0327	806–807	Output vector (F1CA)
0328–0329	808–809	Test–STOP vector (F6ED)
032A–032B	810–811	GET vector (F13E)
032C–032D	812–813	Abort I/O vector (F32F)
032E–032F	814–815	Warm start vector (FE66)
0330–0331	816–817	LOAD link (F4A5)
0332–0333	818–819	SAVE link (F5ED)
033C–03FB	828–1019	Cassette buffer
0340–037E	832–894	(Sprite 13)
0380–03BE	896–958	(Sprite 14)
03C0–03FE	960–1022	(Sprite 15)
0400–07FF	1024–2047	Screen memory
0800–9FFF	2048–40959	Basic RAM memory
8000–9FFF	32768–40959	Alternate: ROM plug–in area
A000–BFFF	40960–49151	ROM: Basic
A000–BFFF	40960–59151	Alternate: RAM
C000–CFFF	49152–53247	RAM memory, including alternate
D000–D02E	53248–53294	Video Chip (6566)
D400–D41C	54272–54300	Sound Chip (6581 SID)
D800–DBFF	55296–56319	Color nybble memory
DC00–DC0F	56320–56335	Interface chip 1, IRQ (6526 CIA)
DD00–DD0F	56576–56591	Interface chip 2, NMI (6526 CIA)
D000–DFFF	53248–53294	Alternate: Character set
E000–FFFF	57344–65535	ROM: Operating System
E000–FFFF	57344–65535	Alternate: RAM
FF81–FFF5	65409–65525	Jump Table, Including:
FFC6		– Set Input channel
FFC9		– Set Output channel
FFCC		– Restore default I/O channels
FFCF		– INPUT
FFD2		– PRINT
FFE1		– Test Stop key
FFE4		– GET

Commodore 64 – ROM Memory Map

A000:	ROM control vectors
A00C:	Keyword action vectors
A052:	Function vectors
A080:	Operator vectors
A09E:	Keywords
A19E:	Error messages
A328:	Error message vectors
A365:	Misc messages
A38A:	Scan stack for FOR/GOSUB
A3B8:	Move memory
A3F8:	Check stack depth
A408:	Check memory space
A435:	'out of memory'
A437:	Error routine
A469:	BREAK entry
A474:	'ready.'
A480:	Ready for Basic
A49C:	Handle new line
A533:	Re–chain lines
A560:	Receive input line
A579:	Crunch tokens
A613:	Find Basic line
A642:	Perform [NEW]
A65E:	Perform [CLR]
A68E:	Back up text pointer
A69C:	Perform [LIST]
A742:	Perform [FOR]
A7ED:	Execute statement
A81D:	Perform [RESTORE]
A82C:	Break
A82F:	Perform [STOP]
A831:	Perform [END]
A857:	Perform [CONT]
A871:	Perform [RUN]
A883:	Perform [GOSUB]
A8A0:	Perform [GOTO]
A8D2:	Perform [RETURN]
A8F8:	Perform [DATA]
A906:	Scan for next statement
A928:	Perform [IF]
A93B:	Perform [REM]
A948:	Perform [ON]
A96B:	Get fixed point number
A9A5:	Perform [LET]
AA80:	Perform [PRINT#]
AA86:	Perform [CMD]
AAA0:	Perform [PRINT]
AB1E:	Print string from (y,a)
AB3B:	Print format character
AB4D:	Bad input routine
AB78:	Perform [GET]
ABA5:	Perform [INPUT#]
ABBF:	Perform [INPUT]
ABF9:	Prompt & input
AC06:	Perform [READ]
ACFC:	Input error messages
AD1E:	Perform [NEXT]
AD78:	Type match check
AD9E:	Evaluate expression
AEA8:	Constant – pi
AEF1:	Evaluate within brackets
AEF7:	')'
AEFF:	comma..
AF08:	Syntax error
AF14:	Check range
AF28:	Search for variable
AFA7:	Setup FN reference
AFE6:	Perform [OR]
AFE9:	Perform [AND]
B016:	Compare
B081:	Perform [DIM]
B08B:	Locate variable
B113:	Check alphabetic
B11D:	Create variable
B194:	Array pointer subrtine
B1A5:	Value 32768
B1B2:	Float–fixed
B1D1:	Set up array
B245:	'bad subscript'
B248:	'illegal quantity'
B34C:	Compute array size
B37D:	Perform [FRE]
B391:	Fix–float
B39E:	Perform [POS]
B3A6:	Check direct
B3B3:	Perform [DEF]
B3E1:	Check fn syntax
B3F4:	Perform [FN]
B465:	Perform [STR$]
B475:	Calculate string vector
B487:	Set up string
B4F4:	Make room for string
B526:	Garbage collection
B5BD:	Check salvageability
B606:	Collect string
B63D:	Concatenate
B67A:	Build string to memory
B6A3:	Discard unwanted string
B6DB:	Clean descriptor stack
B6EC:	Perform [CHR$]
B700:	Perform [LEFT$]
B72C:	Perform [RIGHT$]
B737:	Perform [MID$]
B761:	Pull string parameters
B77C:	Perform [LEN]
B782:	Exit string–mode
B78B:	Perform [ASC]
B79B:	Input byte parameter
B7AD:	Perform [VAL]
B7EB:	Parameters for POKE/WAIT
B7F7:	Float–fixed
B80D:	Perform [PEEK]
B824:	Perform [POKE]
B82D:	Perform [WAIT]
B849:	Add 0.5
B850:	Subtract–from
B853:	Perform [subtract]
B86A:	Perform [add]
B947:	Complement FAC*1
B97E:	'overflow'
B98D:	Multiply by zero byte
B9EA:	Perform [LOG]
BA2B:	Perform [multiply]
BA59:	Multiply–a–bit
BA8C:	Memory to FAC*2
BAB7:	Adjust FAC*1/*2
BAD4:	Underflow/overflow
BAE2:	Multiply by 10
BAF9:	÷ 10 in floating pt
BAFE:	Divide by 10
BB12:	Perform [divide]
BBA2:	Memory to FAC*1
BBC7:	FAC*1 to memory
BBFC:	FAC*2 to FAC*1
BC0C:	FAC*1 to FAC*2
BC1B:	Round FAC*1
BC2B:	Get sign
BC39:	Perform [SGN]
BC58:	Perform [ABS]
BC5B:	Compare FAC*1 to mem
BC9B:	Float–fixed
BCCC:	Perform [int]
BCF3:	String to FAC
BD7E:	Get ascii digit
BDC2:	Print 'IN '
BDCD:	Print line number
BDDD:	Float to asci
BF16:	Decimal constants
BF1A:	TI constants
BF71:	Perform [SQR]
BF7B:	Perform [power]
BFB4:	Perform [negative]
BFED:	Perform [EXP]
E043:	Series eval 1
E059:	Series eval 2
E097:	Perform [RND]
E09:	'?' breakpoints '?'
E12A:	Perform [SYS]
E156:	Perform [SAVE]
E165:	Perform [VERIFY]
E168:	Perform [LOAD]
E1BE:	Perform [OPEN]
E1C7:	Perform [CLOSE]
E1D4:	Parameters for LOAD/SAVE
E206:	Check default parameters
E20E:	Check for comma
E219:	Parameters for open/close
E264:	Perform [COS]
E268:	Perform [SIN]
E2b4:	Perform [TAN]
E30E:	Perform [ATN]
E37B:	Warm restart
E394:	Initialize
E3A2:	CHRGET for zero page
E3BF:	Initialize Basic
E447:	Vectors for $300
E453:	Initialize vectors
E45F:	Power–up message
E500:	Get I/O address
E505:	Get screen size
E50A:	Put/get row/column
E518:	Initialize I/O
E544:	Clear screen
E566:	Home cursor
E56C:	Set screen pointers
E5A0:	Set I/O defaults
E5B4:	Input from keyboard
E632:	Input from screen
E684:	Quote test
E691:	Setup screen print
E6B6:	Advance cursor
E6ED:	Retreat cursor
E701:	Back into previous line
E716:	Output to screen
E87C:	Go to next line
E891:	Perform <return>
E8A1:	Check line decrement
E8B3:	Check line increment
E8CB:	Set color code
E8DA:	Color code table
E8EA:	Scroll screen
E965:	Open space on screen
E9C8:	Move a screen line
E9E0:	Synchronize color transfer
E9F0:	Set start–of–line
E9FF:	Clear screen line
EA13:	Print to screen
EA24:	Synchronize color pointer
EA31:	Interrupt – clock etc
EA87:	Read keyboard
EB79:	Keyboard select vectors
EB81:	Keyboard 1 – unshifted
EBC2:	Keyboard 2 – shifted
EC03:	Keyboard 3 – 'comm'
EC44:	Graphics/text contrl
EC4F:	Set graphics/text mode
EC78:	Keyboard 4
EC89:	Video chip setup
ECE7:	Shift/run equivalent
ECF0:	Screen ln address low
ED09:	Send 'talk'
ED0C:	Send 'listen'
ED40:	Send to serial bus
EDB2:	Serial timeout
EDB9:	Send listen SA
EDBE:	Clear ATN
EDC7:	Send talk SA
EDCC:	Wait for clock
EDDD:	Send serial deferred
EDEF:	Send 'untalk'
EDFE:	Send 'unlisten'
EE13:	Receive from serial bus
EE85:	Serial clock on
EE8E:	Serial clock off
EE97:	Serial output '1'
EEA0:	Serial output '0'
EEA9:	Get serial in & clock
EEB3:	Delay 1 ms
EEBB:	RS–232 send
EF06:	Send new RS–232 byte
EF2E:	No–DSR error
EF31:	No–CTS error
EF3B:	Disable timer
EF4A:	Compute bit count
EF59:	RS232 receive
EF7E:	Setup to receive
EFC5:	Receive parity error
EFCA:	Receive overflow
EFCD:	Receive break
EFD0:	Framing error
EFE1:	Submit to RS232
F00D:	No–DSR error
F017:	Send to RS232 buffer
F04D:	Input from RS232
F086:	Get from RS232
F0A4:	Check serial bus idle
F0BD:	Messages
F12B:	Print if direct
F13E:	Get..
F14E:	..from RS232
F157:	Input
F199:	Get.. tape/serial/rs232
F1CA:	Output..
F1DD:	..to tape
F20E:	Set input device
F250:	Set output device
F291:	Close file
F30F:	Find file
F31F:	Set file values
F32F:	Abort all files
F333:	Restore default I/O
F34A:	Do file open
F409:	Send SA
F40B:	Open RS232
F49E:	Load program
F5AF:	'searching'
F5C1:	Print filename
F5D2:	'loading/verifying'
F5DD:	Save program
F68F:	Print 'saving'
F69B:	Bump clock
F6BC:	Log PIA key reading
F6DD:	Get time
F6E4:	Set time
F6ED:	Check stop key
F6FB:	Output error messages
F72D:	Find any tape headr
F76A:	Write tape header
F7D0:	Get buffer address
F7D7:	Set buffer start/end pointers
F7EA:	Find specific header
F80D:	Bump tape pointer
F817:	'press play.'
F82E:	Check tape status
F838:	'press record..'
F841:	Initiate tape read
F864:	Initiate tape write
F875:	Common tape code
F8D0:	Check tape stop
F8E2:	Set read timing
F92C:	Read tape bits
FA60:	Store tape chars
FB8E:	Reset pointer
FB97:	New character setup
FBA6:	Send transition to tape
FBC8:	Write data to tape
FBCD:	IRQ entry point
FC57:	Write tape leader
FC93:	Restore normal IRQ
FCB8:	Set IRQ vector
FCCA:	Kill tape motor
FCD1:	Check r/w pointer
FCDB:	Bump r/w pointer
FCE2:	Power reset entry
FD02:	Check 8–rom
FD10:	8–rom mask
FD15:	Kernal reset
FD1A:	Kernal move
FD30:	Vectors
FD50:	Initialize system constats
FD8B:	IRQ vectors
FDA3:	Initialize I/O
FDDD:	Enable timer
FDF9:	Save filename data
FE00:	Save file details
FE07:	Get status
FE18:	Flag status
FE1C:	Set status
FE21:	Set timeout
FE25:	Read/set top of memory
FE27:	Read top of memory
FE2D:	Set top of memory
FE34:	Read/set bottom of memory
FE43:	NMI entry
FE66:	Warm start
FE86:	Reset IRQ & exit
FEBC:	Interrupt exit
FEC2:	RS–232 timing table
FED6:	NMI RS–232 in
FF07:	NMI RS–232 out
FF43:	Fake IRQ
FF48:	IRQ entry
FF81:	Jumbo jump table
FFFA:	Hardware vectors

CIA 1 (IRQ) (6526)

$DC00	Paddle Sel A	Paddle Sel B	Joystick 0 Fire	Joystick 0 Right	Joystick 0 Left	Joystick 0 Down	Joystick 0 Up	PRA	56320
	Keyboard Row Select (inverted)								
$DC01	Joystick 1 Fire	Joystick 1 Right	Joystick 1 Left	Joystick 1 Down	Joystick 1 Up			PRB	56321
	Keyboard Column Read								
$DC02	$FF – All Output							DDRA	56322
$DC03	$00 – All Input							DDRB	56323
$DC04	Timer A							TAL	56324
$DC05								TAH	56325
$DC06	Timer B							TBL	56326
$DC07								TBH	56327
$DC0D		Tape Input			Timer Interrupt B	Timer Interrupt A		ICR	56333
$DC0E			One Shot	Out Mode	Time PB6 Out	Timer A Start		CRA	56334
$DC0F			One Shot	Out Mode	Time PB7 Out	Timer B Start		CRB	56335

CIA 2 (NMI) (6526)

$DD00	Serial IN	Clock IN	Serial OUT	Clock OUT	ATN OUT	RS-232 OUT	VIC II addr 15	VIC II addr 14	PRA	56576
$DD01	DSR IN	CTS IN		DCD* IN	RI* IN	DTR OUT	RTS OUT	RS-232 IN	PRB	56577
$DD02	$3F – Serial								DDRA	56578
$DD03	$00 – P.U.P. All Input			or	$06 – RS-232				DDRB	56579
$DD04	Timer A								TAL	56580
$DD05									TAH	56581
$DD06	Timer B								TBL	56582
$DD07									TBH	56583
$DD0D			RS-232 IN				Timer Interrupt B	Timer Interrupt A	ICR	56589
$DD0E								Timer A Start	CRA	56590
$DD0F								Timer B Start	CRB	56591

* Connected but not used by O.S.

Processor I/O Port (6510)

$0000	IN	IN	OUT	IN	OUT	OUT	OUT	OUT	DDR	0
$0001			Tape Motor	Tape Sense	Tape Write	D-ROM Switch	EF RAM Switch	AB RAM Switch	PR	1

SID (6581)

Voice 1	Voice 2	Voice 3		Voice 1	Voice 2	Voice 3
$D400	$D407	$D40E	Frequency L	54272	54279	54286
$D401	$D408	$D40F	Frequency H	54273	54280	54287
$D402	$D409	$D410	Pulse Width L	54274	54281	54288
$D403	$D40A	$D411	0 0 0 0 Pulse Width H	54275	54282	54289
$D404	$D40B	$D412	NSE PUL SAW TRI Voice Type Key	54276	54283	54290
$D405	$D40C	$D413	Attack Time 2ms – 8ms / Decay Time 6ms – 24 sec	54277	54284	54291
$D406	$D40D	$D414	Sustain Level / Release Time 6ms – 24 sec	54278	54285	54292

Voices (write only)

$D415	0 0 0 0 0 Filter Frequency L	54293
$D416	Filter Frequency H	54294
$D417	Resonance / Filter Ext / Filter Voices V3 V2 V1	54295
$D418	V3 off / Passband: HI BP LO / Master Volume	54296

Filter & Volume (write only)

$D419	Paddle X (A/D #1)	54297
$D41A	Paddle Y (A/D #2)	54298
$D41B	Noise 3 (random)	54299
$D41C	Envelope 3	54300

Sense (read only)

Note: Special Voice Features
(TEST, RING MOD, SYNC)
are omitted from the above diagram.

Superscript

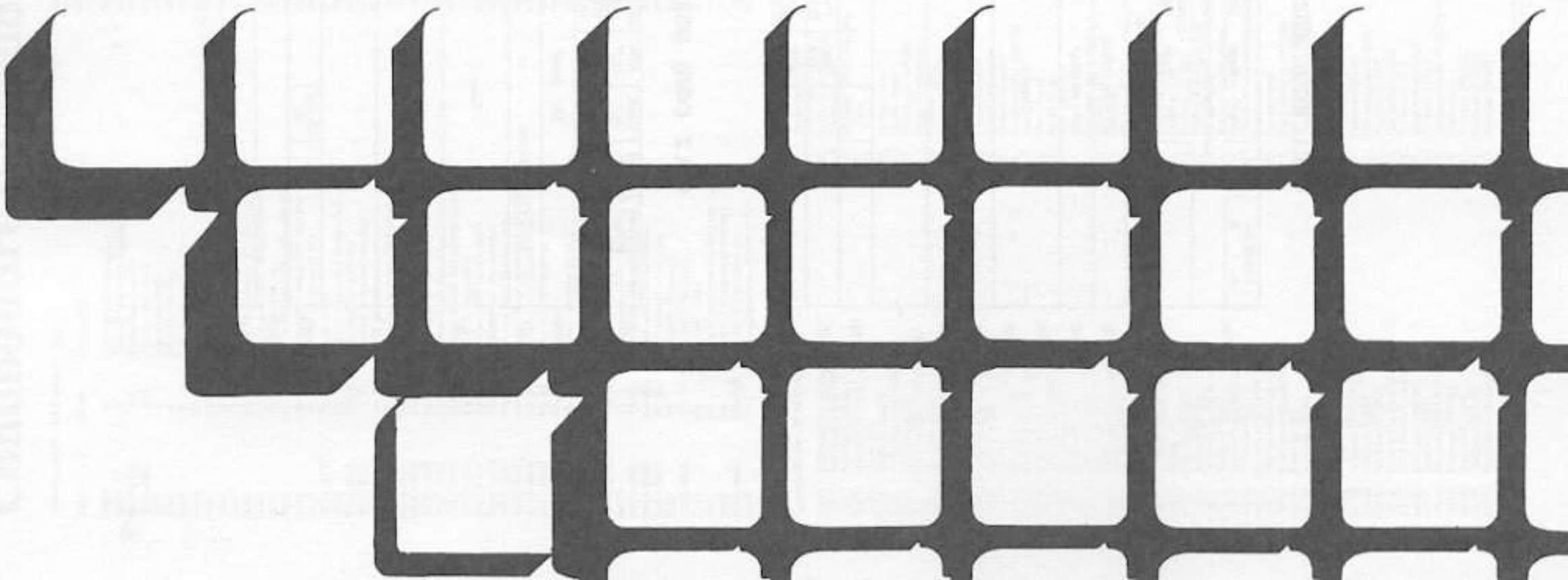
The Ultimate CBM™ Word Processor

A Commodore enthusiast wanted a word processor that was simple, fast and easy to use. He wanted to handle up to 20,000 characters of text, use a wide screen format of up to 240 characters with full window scrolling in all directions and be able to use the screen while printing. He wanted a word processor at a reasonable price. The enthusiast, Simon Tranmer, couldn't find one . . .

So he wrote Superscript

Superscript does everything he wanted. . . and much more. It provides a complete document preparation and storage system, making optimum use of memory and disk space. It gives access to all the letter quality printer features such as boldface and ribbon colour change. In short, it provides all of the advantages of a dedicated professional word processor.

Superscript does everything Commodore wanted. . . which is why they are adopting it for all of their forthcoming models.



ABOUT SUPERSCRIPT

Superscript transforms your Commodore computer into a true word processor, enabling your secretary to turn out high quality letters, mailshots, quotations, contracts and reports more quickly and easily than ever before.

QUALITY AT YOUR FINGERTIPS

Word processing with Superscript allows you to make all the changes you need at the screen, from a simple spelling correction to a complex re-ordering of paragraphs. You can control the final format of the printed document by inserting printer commands where required. These commands include bold face, underlining, superscripts, subscripts, variable pitch and ribbon colour change, in fact all of the attractive formatting features found on modern letter quality printers.

PERFECT COPIES EVERY TIME

Once the preparation of the document is basically complete, you can view it on the screen exactly as it will appear on paper. Then you can either continue to make corrections and improvements, or proceed to print. Superscript will print as many final copies as you like, whenever you like.

UNBEATABLE VALUE

No ROM or cassette key needed
Multi-user at single price

Exclusive Canadian Distributor

$150.00

CMD

CANADIAN MICRO DISTRIBUTORS LTD.
500 Steeles Ave., Milton, Ontario, Canada L9T 3P7/416-878-7277

Superscript

PRIMARY FEATURES

Use of screen as a window on text with **SCROLLING** in all directions
MEMORY SPACE for 250 lines of text 80 columns wide (20,000 characters)
TEXT WIDTHS from screen size up to 240 characters
SIMPLE VIEWING of text with or without embedded commands
Ability to insert into the text explanatory **COMMENTS** that are not printed
LOAD files created by other packages, including Wordcraft™, Wordpro™ and The Manager™, for merging, editing and printing.

EDITING FACILITIES

Ability to execute **COMMANDS** on a succession of linked documents
SEARCH and **REPLACE** of a piece of text in one or a string of documents
Powerful **INSERTION** and **TRANSFER** capability
ERASE all, remainder paragraph or sentence
Automatic or manual **MERGING** of files e.g. for mailing or credit control
HORIZONTAL, VERTICAL and **DECIMAL** tabs, which can be saved on disk
CENTERING and **RIGHT ALIGNMENT** with mix of normal and enlarged characters
HEADERS and **FOOTERS** on every page with auto page numbering

PRINTING

CONTROL of: both margins, lines per page, text lines per page, line spacing, auto line feed and forced paging
DOCUMENT PRINTING: continuous, singlesheet or multicopy singly or as a string of documents with merging as required
BACKGROUND PRINTING: single page or continuous, freeing screen for other tasks
RIGHT JUSTIFICATION with equal white spacing on letter quality printers
PRINT FEATURES include: underlining, enhancement, bold print, superscripts, subscripts, ribbon colour change, variable line and character pitch
SPECIAL PRINTER CHARACTERS can be used e.g. backspace, escape codes, user defined characters

SUPERSCRIPT – one disk runs on the 2001, 4016/32, 8032 and 8096 Commodore computers, 4040 and 8050 disk drives, all Commodore printers and a wide range of letter quality printers.

CBM is a registered trademark of Commodore Business Machines
Wordcraft is a registered trademark of Dataview Limited
Wordpro is a registered trademark of Professional Software Inc.

Advertising Index

Software

Advertiser	Product Name (Description)	Manufacturer	Issue# / Page 1	2	3	4	5	6
Micro Applications	Master (programming aid)		IBC					
Precision Software	Superscript (wordprocessor)		61					

Accessories

Advertiser	Product Name (Description)	Manufacturer	Issue# / Page 1	2	3	4	5	6
Consultors Int'l	Stock Market With Your PC (book)		64					
"	A To Z Book Of Games		64					
"	Dynamics Of Money Mgmt (book)		64					
"	Invment. Analysis w/Your Micro (book)		64					
Leading Edge Inc	Elephant Diskettes		BC					

In The Next Issue Of

The Transactor

- Butterfield Centerfold
- Meet SID – Voice Of The 64
- Commodore 64 Organ Program
- 4040 Power/Error Indicator
- Machine Code Editor
  Link To BASIC–Aid
- Plus More On – VIC–20
  – PET/CBM
  – SuperPET

Keep On Transactin'...

USE YOUR PERSONAL COMPUTER!

Magic! FOR FUN AND PROFIT!

☐ YOU CAN PLAY THE STOCK & BOND MARKETS WITH YOUR PERSONAL COMPUTER! *New! One-of-a-kind 308 page illustrated Source Book can teach you how to use computers in the Investment Field. For Novice or Advanced Computer Hobbyists who want MORE from their machines and make money! Shows how to get started in Stock & Bond Markets* **PLUS** *how to use your computer as a data source for determining profitable stock selections, buy-sell decisions, market evaluations: Order* **#T1251** *Delivered $11.75*

$$$$$$$$$$$$$$$ CHECK*CLIP*MAIL TODAY! $$$$$$$$$$$$$$$$

CLIP THIS AD!

☐ NEW! A TO Z BOOK OF COMPUTER GAMES! HOME 'N' OFFICE FUN-TO-MAKE PROJECTS! *Tested, Ready-to-Run Programs with Adult & Kids Variations, Poker, Black Jack, Roulette,* **PLUS** *Gunners, Knights, Hot-Shot 'N' More! 308 Pages 73 Illust. Order* **T1026** *Delivered $9.75.*

☐ "DYNAMICS OF PERSONAL MONEY MANAGEMENT" The Best Solid Money-Managing Advice ever Stuffed into One Great Source! This is the Prime Source of the Best Money Saving, Money Managing Information Available. A Financial Survival Manual of Saving & Investment Strategies That Will Send Your Net Worth Soaring. Order #4P.. $14.95 Delivered. OUT U.S. by Air $19.95. Canada, Mexico by AIR $19.25

☐ "INVESTMENT ANALYSIS WITH YOUR MICROCOMPUTERS" Latest In Depth Guide using every facet of good Investment Theory & Computer Usage. To Aid in Decision Making Process & Ways to Use The Results to Make Evaluations Quickly. Order #T1479 $12.75 Delivered. OUT U.S. by AIR $16.25. Canada, Mexico by AIR $15.25

(Business Books Tax Deductible)

MONEY BACK GUARANTEE

CHECK YOUR SELECTION ABOVE!

Check M.O. ☐ *VISA* ☐ *MASTERCARD*

_____ *Exp.* _____

Name _____

Address _____

City _____ *State* _____ *Zip* _____

CONSULTORS INT'L. BOX 6589 -D
DENVER, CO 80206 USA

THE ULTIMATE TOOL FOR THE PROFESSIONAL PROGRAMMER

MASTER

MASTER has 70 commands that allow you to create your own business software

MULTIPLE FILE ACCESS & CREATION
Up to 10 ISAM files open simultaneously
Maximum file size not limited by DOS
Supports data packing

SCREEN FORMATTING
Controlled data input
Complete PRINT USING
Graphics supported

REPORT GENERATOR
Multiple file capability
Full printed formatting

96k MEMORY MANAGEMENT
For 8096 or Z-RAM board
56k program space, 26k variables
Up to 16 programs co-resident with common subroutines
Full programmers aid instructions including plotting

BCD MATH PACKAGE
22 digit precision
No binary rounding error

COMPUTED GOTOS AND GOSUBS

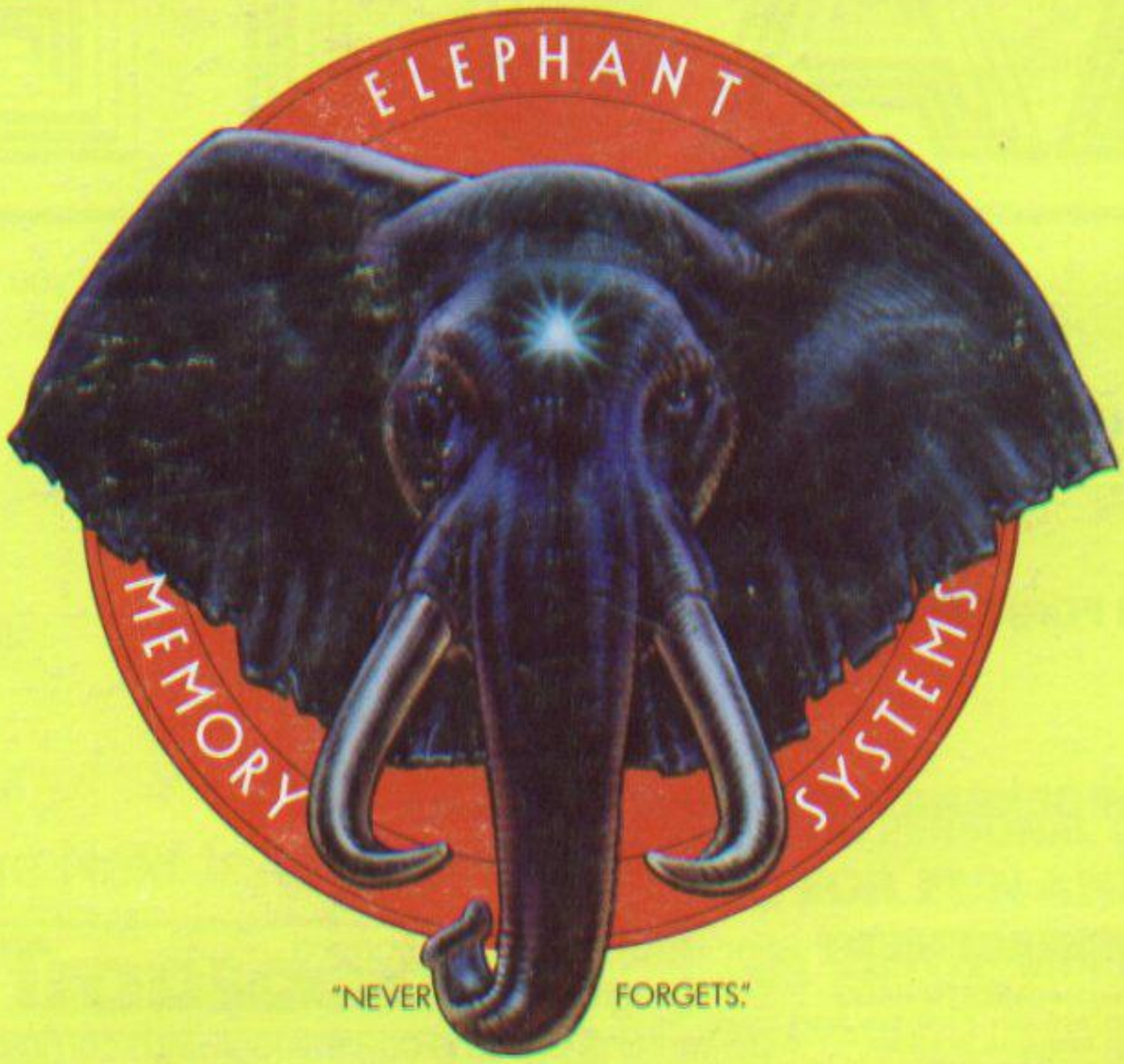
PROGRAM SECURITY WITH DATA-KEY AND NO-LIST OPTION
Now available for:
8096, Z-RAM 8032, 8032 – (no memory management)
Subset for 4032
As a DEVELOPMENT or RUN-TIME system

Developed by MICRO/APPLICATIONS
8096 is Commodore's, Z-RAM is Madison Computers

$350.00
Manual & Demo Disk only for Evaluation $30.00
(Applicable to Purchase of Package)

Distributed by CMD CANADIAN MICRO DISTRIBUTORS LTD.
500 Steeles Ave., Milton, Ontario, Canada L9T 3P7 / 416-878-7277

REMEMBER:



MORE THAN JUST ANOTHER PRETTY FACE.

Says who? Says ANSI.

Specifically, subcommittee X3B8 of the American National Standards Institute (ANSI) says so. The fact is all Elephant™ floppies meet or exceed the specs required to meet or exceed all their standards.

But just who is "subcommittee X3B8" to issue such pronouncements?

They're a group of people representing a large, well-balanced cross section of disciplines—from academia, government agencies, and the computer industry. People from places like IBM, Hewlett-Packard, 3M, Lawrence Livermore Labs, The U.S. Department of Defense, Honeywell and The Association of Computer Programmers and Analysts. In short, it's a bunch of high-caliber nitpickers whose mission, it seems, in order to make better disks for consumers, is also to make life miserable for everyone in the disk-making business.

How? By gathering together periodically (often, one suspects, under the full moon) to concoct more and more rules to increase the quality of flexible disks. Their most recent rule book runs over 20 single-spaced pages—listing, and insisting upon—hundreds upon hundreds of standards a disk must meet in order to be blessed by ANSI. (And thereby be taken seriously by people who take disks seriously.)

In fact, if you'd like a copy of this formidable document, for free, just let us know and we'll send you one. Because once you know what it takes to make an Elephant for ANSI . . .

We think you'll want us to make some Elephants for you.

ELEPHANT.™ HEAVY DUTY DISKS.

For a free poster-size portrait of our powerful pachyderm, please write us.
Distributed Exclusively by **Leading Edge Products, Inc.**, 225 Turnpike Street, Canton, Massachusetts 02021
Call: toll-free 1-800-343-6833; or in Massachusetts call collect (617) 828-8150. Telex 951-624.